Design, Automation and Test for Asynchronous Circuits and Systems

Information Society Technologies (IST) Programme
Concerted Action Thematic Network Contract
IST-1999-29119

D. A. Edwards W. B. Toms

3rd Edition June 2004

Contents of Report

Introduction

This report forms part of the deliverable for workpackage 3 of contract IST-1999-29119. It is intended to describe the state-of-the-art in methods and tools for the design of asynchronous digital VLSI systems. This report is intended to be primarily of use to companies, both members and non-members of the Working Group on Asynchronous Circuit Design (*http://www.bcim.lsbu.ac.uk/ccsv/ACiD-WG/*), who are aware of the potential benefits of asynchronous circuit technology, but who need to know more about available asynchronous design methods and tools before committing resources. The report aims to highlight deficiencies in existing approaches and so provide the impetus for further tools development.

Notes on the 3rd Edition:

Nine new tools have been described: CAST, DGC, Oolong, Phased Logic, Verimap, VerySAT and Weaver are described in their own right; syndi and diana are included under the di2pn tool description. Tangram has been replaced by Haste and is now commerically available. Updated submissions were received about the tools ATACS, Balsa, Cascade, Clp, ConfRes, Desi, di2pn, Optimist, Pipefitter, Punf and Theseus NCL Synthesis Flow. No further information has been received about the tools 3D, ACK, CADP, FIREMAPS, SIS, Transyt, Veraci, VSTGL. It is assumed they are in the same state as last year and so their previous submissions have been kept. New references have been added to the Test section.

Notes on the 2nd Edition:

Several organisations have updated their descriptions with new features, FIREMAPS, CADP and di2pn all have new descriptions, ATACS now has a wider availability. It is expected that there are changes to entries for Minimalist and Tangram, but no information is available as yet. There are no notable changes for most of the other tools in the first report. No further information has been received about the tools 3D, ACK, Theseus NCL Flow, Transyt and XDI. It is assumed that these tools are in the same state as last year, and so their previous submissions have been kept. The tool VHDL++ has been removed as it is no longer being maintained. Professor Graham Birtwhistle, who pioneered CCS verification of asynchronous circuits, has retired from his post at the University of Leeds, UK, but currently has a temporary post at the Dept of Computer Science, University of Sheffield, UK. Six new tools have been added: CASCADE, ConfRes, Clp, DESI, Optimist, Punf. New references have been added to the Test section.

Introduction to the 1st Edition:

Scope of the Report

The tools in this report have not been assessed by the authors. Rather, they are a self-evaluation by the creators of the tools in response to an email questionnaire. A search of the published literature has been used to identify tool developers in areas relevant to asynchronous circuits.

Structure of the Report

The main body of the report contains the self evaluations received from various tools developers. Test methodologies for asynchronous circuits are still in their infancy. The section “Testing Asynchronous Circuits” details some of the more important papers in this area. Appendix A, “Identified Tools & Methodologies,” on page 102 of the report identifies tool and methodology developers who were contacted, The original questionnaire is reproduced in Appendix B, “Original Questionnaire,” on page 118.

In total, twenty two tools/methodologies are listed in alphabetical order. Four of these, (ACK, Balsa, Tangram and TAST) are primarily concerned with silicon compilation, and two (LARD and VHDL++) with simulation. Two (BUTLER and NCL technologies) exploit libraries of special-purpose components. Five (3D, ATACS, MINIMALIST, Petrify and SIS) perform logic synthesis, with three more (di2pn, Pipefitter and VSTGL) usable as front ends to Petrify. Finally, six (CADP, CCS, Firemaps, Transyst, Veraci and XDI) support formal verification.

Tool/Methodology: 3D

Developer: Ken Yun

Organisation: University of California, San Diego

Summary

Today's system components typically employ the synchronous paradigm primarily because of the availability of the rich set of design tools and algorithms and, perhaps, because of the designers' perception of "ease of design" and the lack of alternatives. Even so, the interfaces among the system components do not strictly adhere to the synchronous paradigm because of the cost benefit of mixing modules operating at different clock rates and modules with asynchronous interfaces.

In order to tackle this problem, the synthesis system called 3D has been developed. The 3D system uses a design style called extended burst-mode (XBM). The XBM design style covers a wide spectrum of sequential circuits ranging from delay-insensitive to synchronous. It can synthesize multiple-input change asynchronous finite state machines, and many circuits that fall in the gray area between synchronous and asynchronous which are difficult or impossible to synthesize automatically using existing methods. This implementation of XBM machines uses standard combinational logic, generates low-latency outputs and guarantees freedom from hazards at the gate level.

Strengths and Weaknesses

Not stated.

Application domain

Controllers operating in heterogeneous systems – systems with components employing different synchronization mechanisms.

Use of Existing HDLs

Currently, the 3D system is integrated with HFMIN, an exact two-level logic minimizer, implemented by Robert Fuhrer and Steve Nowick at Columbia University. The integrated technology mapper is adaptable to most commercial CMOS standard cell and mask programmable gate array libraries. Inputs to the 3D tool are textual descriptions of XBM controllers, and outputs are technology-independent logic equations and technology- mapped netlists.

Extent of Automation

The 3D synthesis system is an implementation of a complete set of automated sequential synthesis algorithms: hazard-free state assignment, hazard-free state minimization, and critical-race-free state encoding.

Category

Synthesis – Extended Burst Mode asynchronous controllers.

Design Flow and Commercial EDA Tool Requirement

Not stated.

Test Strategy

Not stated

Current Status of Tool

Current Activities

The 3D synthesis system is completely functional now. It synthesizes XBM controllers in two-level AND-OR logic and maps them to a generic CMOS standard cell library or in generalized C-elements. All known bugs have been fixed. It has been used to synthesize the above mentioned chips and is actively used by several research groups, including Intel, IBM, and University of Utah VLSI group.

Maintainer

Ken Yun, *kyy@ucsd.edu*

Availability

Available from: *http://paradise.ucsd.edu/3D/*

Future plans

There is an on-going research efforts to generate multi-level logic and generalized C-element based implementations.

Demonstrators

The effectiveness of the XBM design style and the synthesis tool has been demonstrated by designing several chips or modules, some of which are are listed below:

- a low-control-overhead differential equation solver in 0.5um HP CMOS14TB process; *http://paradise.ucsd.edu/3D/examples/diffeq*
- a commercial-scale SCSI controller, which is functionally compatible with an existing high performance commercial chip and meets the ANSI SCSI-2 standard; *http://paradise.ucsd.edu/3D/examples/scsi*
- an infrared communications chip (in collaboration with HP Labs) capable of transferring data at 800Kbps only dissipating 200mW at peak data rate.

References

See http://paradise.ucsd.edu/controller.html for hyptext links to the following publications:

[1] W.-C. Chou, P. A. Beerel, and K. Y. Yun, “Average-case technology mapping of asynchronous burst-mode circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 10, pp. 1418-1434, Oct. 1999.

[2] S. Chakraborty, K. Y. Yun, and D. L. Dill, "Timing analysis of asynchronous systems using time separation of events," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 8, pp. 1061-1076, Aug. 1999.

[3] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: part I (specification and hazard-free implementations)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 101-117, Feb. 1999.

[4] K. Y. Yun and D. L. Dill, "Automatic synthesis of extended burst-mode circuits: part II (automatic synthesis)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 2, pp. 118-132, Feb. 1999.

[5] S. Chakraborty, D. L. Dill, and K. Y. Yun, "Min-max timing analysis and an applications to asynchronous circuits," *Proceedings of the IEEE*, vol. 87, no. 2, pp. 332-346, Feb. 1999.

[6] K. Y. Yun, B. Lin, D. L. Dill, and S. Devadas, "BDD-based synthesis of extended burst-mode controllers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 9, pp. 782-792, Sept. 1998.

[7] S. Chakraborty, K. Y. Yun, and D. L. Dill, "Practical timing analysis of asynchronous systems using time separation of events," in *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference*, Santa Clara, California, May 1998, pp. 455-458.

[8] K. W. James and K. Y. Yun, "Average-case optimized transistor-level technology mapping of extended burst-mode circuits," *in Proceedings of the 1998 International Symposium on Advanced Research in Asynchronous Circuits and Systems*, San Diego, California, Mar. 1998, pp. 70-79.

[9] W. Chou, P. A. Beerel, R. Ginosar, R. Kol, C. J. Myers, S. Rotem, K. Stevens, and K. Y. Yun, "Average-case optimized technology mapping of one-hot domino circuits," in *Proceedings of the 1998 International Symposium on Advanced Research in Asynchronous Circuits and Systems*, San Diego, California, Mar. 1998, pp. 80-91.

[10] S. Chakraborty, D. L. Dill, K. Y. Yun, and K-Y. Chang, "Timing analysis for extended burst-mode circuits," in *Proceedings of the 1997 International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Eindhoven, The Netherlands, Apr. 1997, pp. 101-111.

[11] K. Y. Yun, "Automatic synthesis of extended burst-mode circuits using generalized C-elements," in *Proceedings of the 1996 European Design Automation Conference*, Geneva, Switzerland, Sept. 1996, pp. 290-295.

[12] P. A. Beerel, K. Y. Yun, and W. Chou, "A heuristic covering technique for optimizing average-case delay in the technology mapping of asynchronous burst-mode circuits," in *Proceedings of the 1996 European Design Automation Conference*, Geneva, Switzerland, Sept. 1996, pp. 284-289.

Tool/Methodology: ACK

Developers: Hans Jacobson, Prabhakar Kudva, Ganesh Gopalakrishnan, Erik Brunvand

Organisation: University of Utah

Summary

We define asynchronous systems to be medium to large digital systems whose descriptions include both datapath and control, that may involve non-trivial interface requirements, and whose control is too large to be synthesized in one large controller. ACK is a framework for designing high-performance asynchronous systems of this type. In ACK we advocate an approach that begins with procedural level descriptions of control and datapath and results in a hybrid system that mixes a variety of hardware implementation styles including burst-mode AFSMs, macromodule circuits, and programmable control. Specifically, ACK is a high level synthesis tool that describes the desired system at a procedural level (including datapath specification), and automatically compiles that specification into interconnected control and datapath circuits.

Apart from creating an automated path from high-level specification all the way down to layout, our work on ACK so far has concentrated on providing what we consider to be three of the most important features of a successful asynchronous high-level synthesis tool. While many additional features are needed to form a complete tool these are what we consider to be the most important basic building blocks of the type of framework for asynchronous system synthesis we have outlined.

- New efficient control structures for asynchronous hardwired as well as programmable control. These structures also support thread-level concurrency and sequential chaining of system tasks.
- Provide high-level design optimisation with comprehensible feedback.
- Partitioning and fast and exact synthesis of ASFMs to enable exploration of large design spaces.

Strengths and Weaknesses

See previous bulleted list. Enables high level design by allowing system level descriptions that include both control and datapath. Weaknesses are that the tools backend isn’t currently working as we switch back-end tool support from Cascade to Cadence, no fancy interface so the tool is difficult to drive, some parts of the flow are integrated, others are separate point tools at the moment.

Application domain

Medium to large asynchronous digital systems whose descriptions include both datapath and control, that may involve non-trivial interface requirements, and whose control is too large to be synthesized in one large controller.

Use of Existing HDLs

Front end uses standard Verilog simulation tools (Veriwell, Verilog-XL, etc.), Datapath synthesis uses Synopsys, complex gate synthesis uses Cadence, back-end assembly worked using Cascade, which is no longer available. We are switching to Cadence.

Category

Synthesis – Partitioned Burst Mode or timed async controllers, standard synchronous datapath synthesis.

Design Flow and Commercial EDA Tool Requirement

see figure.

Current Status of Tool

In current development. Entire tool is not ready for wide release.

Maintainer

Hans Jacobson, hans@cs.utah.edu

Future plans

Integrate into more usable interface, add back-end support from Cadence, add microengine controller support, firm up Verilog front end.

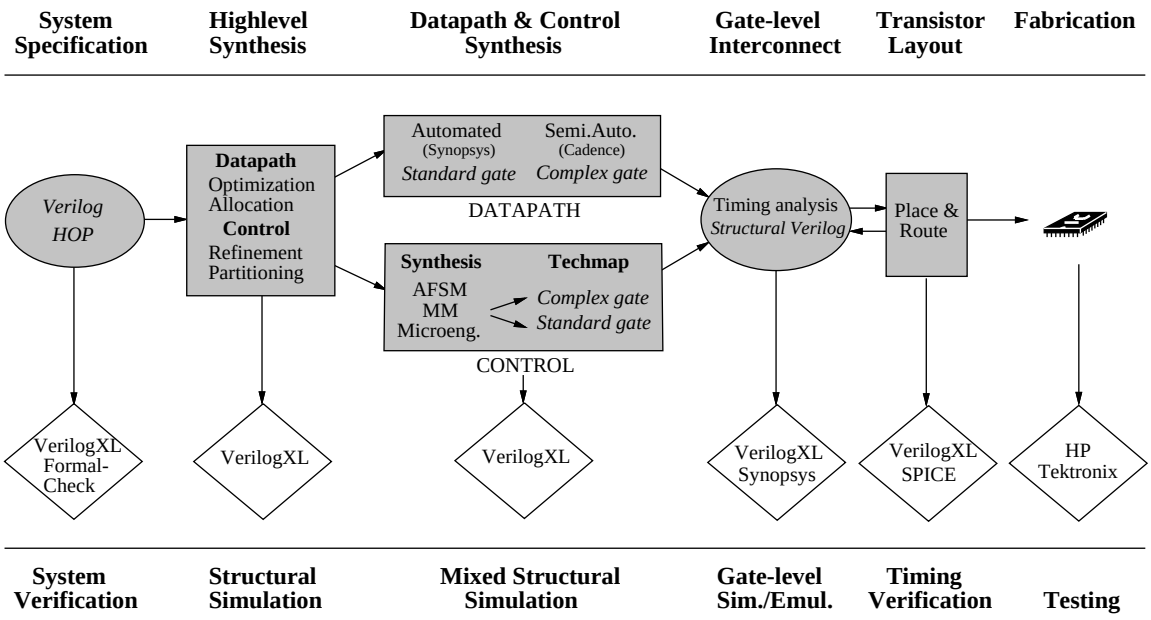

Figure 1: ACK Tool Flow

Tool/Methodology: ATACS (Automated Timed Asynchronous Circuit Synthesis)

Developers: Chris Myers, Wendy Belluomini, Hans Jacobson, Sung-Tae Jung, Chris Kreiger, Scott Little, Eric Mercer, Eric Peskin, Nick Seegmiller, Robert Thacker, David Walter, Hao Zheng

Organisation: University of Utah, USA

Summary

ATACS is a synthesis, analysis, and verification tool for timed circuits. Timed circuits are defined to be any circuits that are optimised using explicit timing information. One example is the self-resetting and delayed-reset domino circuits used in IBM's gigahertz research microprocessor. Much of the improvement in speed in this processor can be attributed to these aggressive circuit styles. Another example is the asynchronous circuits utilized by Intel's RAPPID instruction length decoder. This design was 3 times faster while using only half the power of the comparable synchronous design. These results were obtained using aggressive timing assumptions.

The ATACS tool accepts designs given VHDL, handshaking expansions, Petri-nets, burst-mode state machines, timed event/level structures, and state graphs. VHDL descriptions can model a system either at a communication channel level or a signal transition level. Simulation at both levels is supported by VHDL packages developed in our group, and the VHDL code using these packages can be simulated using any commercial VHDL simulator. In most of the specification methods, timing annotations can be made to indicate knowledge about the delays of any given signal transition in the design. This delay information is utilized throughout the design process to optimise the design which is produced. Each of these input forms is compiled into an internal graphical representation. At this point, one of numerous timing analysis methods can be selected to find only those states which are reachable given the provided timing information. Automatic abstraction and partial order methods are provided to improve the efficiency of timed state space exploration.

After finding the reachable state space, a timed circuit implementation can be synthesized. ATACS supports several synthesis algorithms including efficient explicit state methods, BDD implicit methods, and a direct synthesis method which avoids state space exploration and simply generates the circuit from a free-choice Petri net representation. Each synthesis method generates circuits which are hazard-free under a particular technology model. The choices of technology model are: atomic gate, generalized C-elements, standard C-elements, and burst-mode.

ATACS also supports analysis and verification. For analysis, a stochastic simulation is performed utilizing provided delay distributions. The result is a reported average-case performance as well as detailed information about areas of the design which contribute the most to the cycle time. For verification, a designer can provide timing constraints that should be checked during timed state space exploration. If a circuit is provided, the circuit will be also be checked for hazard-freedom. Finally, several other properties are checked during state space

exploration including net safety and deadlock. When errors are found a graphical error trace is provided.

Strengths and Weaknesses

Allows for the systematic design of extremely efficient and aggressive circuits. Application is limited to moderate size blocks of a design.

Application Domain

High performance circuits.

Use of Existing HDLs

Utilizes VHDL as both a front end and back end allowing simulation and integration with commercial tools.

Extent of Automation

Fully automated approach.

Category

Synthesis => timing driven

Design Flow and Commercial EDA Tool Requirement

A VHDL simulator is useful. Physical design tools are needed to realize the synthesized logic.

Test Strategy

ATACS supports performance analysis and formal verification. Functional simulation is performed using a VHDL simulator. There is no support for test generation.

Current Status

Current Activities

ATACS is in continuous development.

Maintainer

Chris Myers (myers@ee.utah.edu)

Availability

ATACS is available to download for free for academic institutions, SRC member companies, and government agencies for non-commercial research purposes from:

http://www.async.ece.utah.edu.

Others may license ATACS from the University of Utah Technology Transfer Office.

Future Plans

Improve integration with existing design flows.

Demonstrators

ATACS was used during the Intel RAPPID project (see JSSC 36(2): 217-228). It was also used to verify several circuits from GUTS (see TCAD 20(1):129-146).

Tool/Methodology: Balsa

Developers: Andrew Bardsley, Doug Edwards, Lilian Janin

Organisation: University of Manchester, UK

Summary

The Balsa System is a Handshake Circuit based, macromodule synthesis tool-set. Design descriptions are written in the proprietary language Balsa and synthesised into networks of handshake components in a similar manner to the Philips Tangram compiler. These handshake circuits are expressed in a format called Breeze. Breeze is used by all the Balsa tools as a design repository format making the backend tools independent of the frontend Balsa language. Breeze descriptions can be realised as standard-cell VLSI layout using a combination of Balsa tools and commercial CAD. The Balsa language is based on the synchronous channel communicating, fine grain parallel descriptive style of CSP and shares many features with Tangram and OCCAM. The Balsa system can currently generate circuits for several backend technologies: Xilinx FPGAs, ARM generic design rules and cell library (as used to implement AMULET3), two ASIC cell libraries: a 0.35um from Austria Mikro Systems and the 0.18um HCMOS8D STMicroelectronics. A custom-built cell library using the ST process is also available. Adding technologies is relatively simple and tools are available to allow users to target their own cell-libraries. Three choices of data encodings are possible for each of these technologies: single-rail - based in part on our previous work on the EXACT project; dual-rail and 1-of-4 - for easier timing validation. Two further backends are also currently in development: early-single-rail - to allow overlapping of return-to-zero phases of handshakes, and m-of-n encoded - allowing each channel to adopt arbitrary encodings.

The most recent release (3.4) has greatly improved simulation within the Balsa framework Lard has been replaced with the much more efficient *breeze-sim* which allows source-level debugging of Balsa code. The Lard viewer has been replaced by *gtkwave* a signal-viewer maintained by the University of Manchester - which has been enhanced to display channel communications. An animated Handshake Circuit visulisation tool allows circut allows circuit behaviour to be examined at the Breeze level. The Balsa language has also been updated with new unsynthesisable constructs, such as printing, that aid the construction of test-benches within the Balsa language itself. The balsa-verilog-sim package provides a VPI/PLI interface to several commercially and freely available simulators, allowing complete verilog simulation within the balsa environment.

The Balsa System comprises:

- balsa-c: the Balsa to Breeze compiler.
- balsa-netlist: Breeze to CAD system netlist expansion. Balsa-netlist processes descriptions of the backend technology to produce technology specific netlists.
- breeze-cost: circuit cost estimation for Breeze.
- breeze2ps: Breeze handshake circuit pretty printer.

- balsa-mgr, balsa-md: design management tools. Balsa-mgr is a friendly GUI frontend for the Balsa system allowing design descriptions to be compiled and simulated more easily.
- balsa-make-test: automatically generates test harness for a Balsa description.
- breeze-sim: the preferred simualtor working at the handshake component level.
- breeze-sim-control: a graphical front-end to the simulation and visualisation environment
- balsa-verilog-sim: a package which makes Verilog simulation of Balsa descriptions easier by providing wrapper scripts for common simulators and by supporting user-written builtin functions which can be called from Balsa

Strengths and Weaknesses

Rapid development time, transparent design approach. The design of the Balsa System allows new backend technologies and data encodings/handshake protocols to be easily added allowing the designer to choose a point in the speed/area/power design space.

Possibly not optimally efficient.

Application Domain

High complexity, medium performance circuits.

Use of Existing HDLs

Compass, Cadence, Xilinx and Mentor-Graphics commercial tools are used to implement Balsa designs. Balsa-mgr specifically targets several verilog simulators - Icarus, Cver, Cadence’s NCVerilog and VerilogXL, Synopsys VCS.

Extent of Automation

Push-button approach from description to layout. Balsa-mgr supports generation and simulation of within the framework, from Breeze to Verilog. A complete flow to layout exists for the custom-built Amust018 cell library using the ST process.

Category

Synthesis - Silicon compilers

Design Flow and Commercial EDA Tool Requirement

The current Balsa design flows are shown in Figure 1

Test Strategy

Balsa provides an interface to the Breeze-SIM and several commercial verilog simulators for functional simulation. There are no test vector generation tools.

Current Status of Tool

Current Activities

In current development.

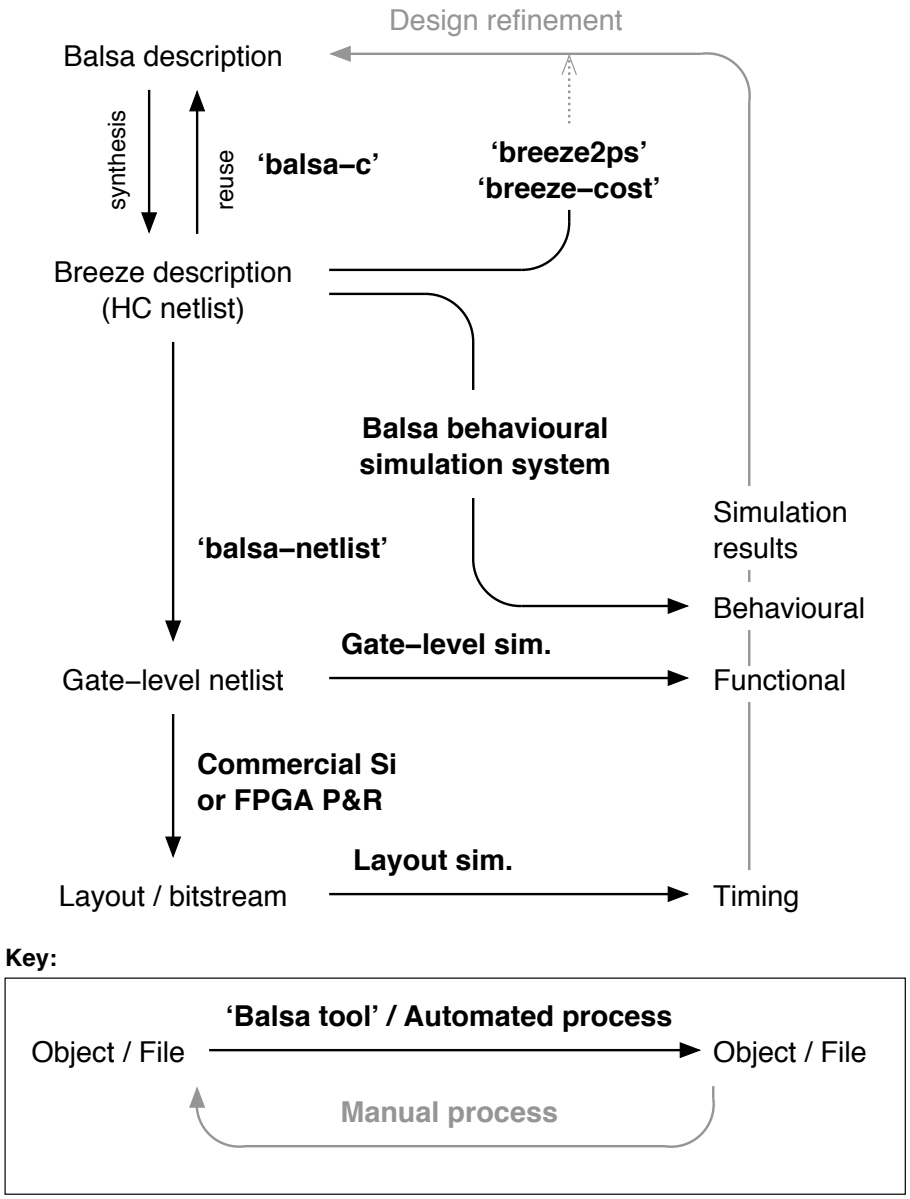

Figure 1: Balsa Design Flows

Maintainer

Contact: *balsa@cs.man.ac.uk*

Availability

GNU GPL

available from: *http://www.cs.man.ac.uk/apt/projects/balsa*

Future plans

Better simulation support, Datapath compilation, More target technologies, Data encoding choices

Demonstrators

AMULET3i DMA controller described in “Synthesising an asynchronous DMA controller with Balsa” Andrew Bardsley, Doug Edwards Journal of Systems Engineering 46 (2000) pp 1309-1319

Tool/Methodology: Butler

Developer: Eric Campbell

Organisation: MBDA

Summary

The Butler Technology provides a general solution to the problem of managing processing and communication resources in an embedded real-time multiple processor system. It includes generic designs for special hardware components that work in conjunction with minimal executive software to provide an efficient execution environment for application functions. The generic designs are parameterised to obtain project specific versions which are then incorporated with conventional processing and communications hardware designs to produce robust, high performance, computing modules.

The generic designs have a modular design structure, being constructed from an assembly of design tiles. Each tile is a design building block that comprises both logic and structure. The logic in each tile is expressed in terms of a few interconnected simple gates. Tiles are butted together to form a two dimensional array that realises overall functionality and in some implementations can directly form part of an integrated circuit layout.

A clock-free, event driven, design approach has been used within the generic designs. This reflects the nature of the reactive computational model being supported. Each design is analysed for correctness by formal mathematical methods. The designs can be easily integrated with other circuitry and implemented in different fabrication technologies because they are not dependent on critical timing parameters. The circuitry has non-demanding power supply requirements.

One of the generic designs is called the butler, a device that can be used with any microprocessor. It provides efficient support for multi-tasking in a single or multiple processor system. It holds control variables for each task assigned to run on the microprocessor and at execution time identifies the next task that should run. The logic for selecting the next task is programmable. Priority levels can be allocated to individual tasks or groups of tasks. Tasks that are given the same priority are selected on a round robin basis within their group. Asynchronous stimuli (e.g. interrupt lines from local peripherals) are handled directly by the butler, scheduling the relevant task when its turn arrives according to the programmed priority level selection. Cooperative and pre-emptive scheduling schemes are supported.

Another of the generic designs is called the route-table, a device that can be used with any physical communication medium. It enables many software routes, each with a range of different interaction characteristics, to be multiplexed onto the same physical communication medium. The butler and route-table operate together to manage the communication and processing resources in a system in a defined way with a minimum of software overhead.

Strengths and Weaknesses

A major advantage of the approach, especially for products with a long in-service life, is that it does not require the use of any special languages or tools. Designs can be re-implemented,

when hardware obsolescence becomes a problem, using the technology and tools in use at the time.

Application Domain

Our problem domain is the development of dependable, small, harsh-environment, high-performance, low-power, embedded computing in guided weapon systems.

Use of Existing HDLs

Extent of Automation

Category

Design Flow and Commercial EDA Tool requirement

The butler technology generic designs are integrated with conventional clocked circuitry, often on the same silicon. We use a conventional synchronous design flow overall but inhibit some stages on the asynchronous parts. E.g. We do not use synthesis and optimization because the generic design is already expressed at simple gate level. We do not need to run timing analysers and insert scan paths because we have no clocked latches.

Test Strategy

We analyse the generic design for correctness by formal mathematical methods (this covers all aspects for all versions). This confirms both that the generic design satisfies its specification and that its specification provides for correct system operation.

Knowing that the design for any version is correct means that our production testing need only confirm that each manufactured item conforms to its design. I.e. that every transistor is present and correct. The test pattern developed for each tile type is simply repeated for each tile instance in the array. The clock-free ripple-through design style allows internal states to be observed externally and we are able to achieve full manufacturing fault coverage

Current Status of Tool

Current Activities

We are applying the butler technology in the design process for a ‘next generation’ computing module for use in a new missile system

Maintainer

Eric Campbell MSc., CEng., M.I.E.E., Technologist Computing Architectures, PB 77, MBDA UK, Six Hills Way, Stevenage SG1 2DA.

eric.campbell@mbda.co.uk

Tel:+44(0)1438 755268 Fax:+44(0)1438 756293

Availability

Future Plans

The butler technology does not require the use of any special asynchronous design tool. The butler technology generic designs are being developed further to include additional features

Demonstrators

The Butler Technology is currently used in two of our missile systems: an advanced short range air-to-air missile called ASRAAM; a ship based anti-missile missile called Seawolf

Tool/Methodology: CADP

Developer: VASY Team

Organisation: INRIA Rhone-Alpes

Summary

CADP (CAESAR/ALDEBARAN Development Package) is a toolbox for protocol engineering. It offers a wide range of functionalities, from interactive simulation to the most recent formal verification techniques. CADP is maintained by the VASY project at INRIA Rhone-Alpes. It is dedicated to the efficient compilation, simulation, formal verification, and testing of descriptions written in the ISO language LOTOS [ISO standard 8807]. In particular, the toolbox include state-of-the-art tools to perform (enumerative, on the fly, compositional) verification using either bisimulation or mu-calculus methodologies.

Strengths and Weaknesses

Clearly, the asynchronous concurrency model of LOTOS, which is based upon interleaving semantics, is appropriate for describing networks of processes that execute in parallel and communicate by message passing. This approach is especially suitable for multiprocessor architectures which are difficult to describe accurately using a synchronous approach. Examples can be found at the following URLs:

http://www.inrialpes.fr/vasy/cadp/case-studies/00-f-circuits.html

http://www.inrialpes.fr/vasy/cadp/case-studies/98-f-async-circuits.html

On the other hand, languages based on automata communicating by FIFO queues are not well adapted to hardware systems, as they do not allow to model instantaneous communications (using electric signals) between hardware components. For instance, the request of a processor wanting to access a bus is better expressed using a LOTOS rendez-vous than by putting a message in an infinite FIFO queue.

Compared to other asynchronous process algebras, LOTOS has the merit of being an established international standard, for which many textbooks and tutorials are available (even on the Internet).

Finally, the CADP tools are robust and widely disseminated in more than 274 sites around the world (data: November 2002). They are also available for several platforms including Sun Solaris, Linux PCs and Windows PCs.

Application Domain

The semantic model of process algebras is general enough not to be tied to a particular kind of hardware systems. For instance, the LOTOS language and CADP tools have also been used to model software and telecommunication systems. So far, in the VASY team of INRIA, we have used LOTOS and the CADP tools to verify bus arbitration and cache coherency protocols for high-end server multiprocessor architectures developed by Bull, the link layer of the IEEE 1394 bus, the bus arbitration protocol of SCSI-2, etc. See:

http://www.inrialpes.fr/vasy/dyade/vasy.html

http://www.inrialpes.fr/vasy/dyade/formalfame.html

http://www.inrialpes.fr/vasy/Press/firewire.html

http://www.inrialpes.fr/vasy/verdon.

Use of Existing HDLs

We operate at system level. At this level, there is no well-established HDL yet, so our industrial partners are ready to adopt process algebraic formalisms provided that they are supported by robust tools that bring concrete results

Extent of Automation

The enumerative approach used in the CADP allows a large degree of automation. However, to make the best use of the tools, it is suitable to use trained professionals with a background in formal methods and verification tools.

Category

Formal verifiers/Theorem provers

Design Flow and Commercial EDA Tool Requirement

As far as hardware design is concerned, our methodology consists in establishing a reference specification of the system under design using a formal description technique such as LOTOS. We usually start from an informal description of the system, which we turn into a formal one. So doing, inconsistencies and uncovered issues are often detected. Then, we use the CADP tools to simulate, validate and verify the formal description. This work allows the informal description to be refined and corrected. Then, the informal and formal specifications are used by code writers as a basis to develop Verilog or VHDL code. Additionally, we can use the formal description to generate or validate test suites (see below).

Test Strategy

The CADP toolbox can be used for generating test suites automatically using the TGV tool contained in the CADP distribution. The test suites are derived from a reference specification of the system using a formal description technique (such as LOTOS). They are used to assess the correctness of the actual implementation.

The CADP toolbox can also be used in conjunction with the TorX architecture, developed jointly by the University of Twente, the University of Eindhoven, and Philips Research. TorX is a flexible and open architecture that allows on-the-fly testing, batch test derivation and batch test execution for different specification formalisms.

The CADP toolbox can also be used to check the correctness of execution traces obtained from the real system (or from an execution of Verilog or VHDL code). These traces can be produced using either random testing or “focused” testing. More often, such traces are tedious to verify by a human. The CADP tools allow to check automatically if these traces are accepted by the formal specification.

Current Status of Tool

Current activities

The CADP toolbox is distributed, maintained and improved by the VASY team of INRIA. Other research groups contribute actively to the development of CADP, and especially the Verimag laboratory, the PAMPA team of INRIA/IRISA, and the TIOS team at the University of Twente.

Maintainer

See above.

Availability

Information regarding the CADP toolbox, its availability, its recent changes and improvements is available from the following URL*:*

http://www.inrialpes.fr/vasy/cadp.

This Web page contains centralized up to date information regarding CADP

Future Plans

Significant Demonstrators

Two published papers regarding the application of CADP to system-level design of multiprocessor architectures:

http://www.inrialpes.fr/vasy/Publications/Chehaibar-Garavel-et-al-96.html

http://www.inrialpes.fr/vasy/Publications/Garavel-Viho-Zendri-00.html

References

A list of published case-studies involving CADP is available from

http://www.inrialpes.fr/vasy/cadp/case-studies

A list of third-party software developed using CADP is available from

http://www.inrialpes.fr/vasy/cadp/software

Tool/Methodology: CASCADE

Developer: CASCADE-Team (Chair of Digital Technic)

Organisation: Kaiserslautern University of Technology

Summary

CASCADE (Communicating Asynchronous Sequential Circuits: Architecture Development Environment) is a hardware design tool that supports a comprehensive Petri net based design method for asynchronous controllers. Currently, a designer wishing to synthesize a controller consisting of one or several communicating asynchronous circuits is faced with the problem of first having to choose an appropriate design style and then formulating the design problem in that style's particular specification scheme (which in general is not a Petri net). A better approach would be to

1. start from a unified design entry using a specification scheme capable of expressing every known kind of asynchronous controller behaviour, and
2. then decide upon the appropriate synthesis method.

Asynchronous circuits are event-driven. It would be adequate, therefore, to specify the required input-output behaviour from a causal point of view. A causal specification scheme already in use is the signal transition graph (STG) [8]. STGs are interpreted place-transition Petri nets where the firing of a transition represents the occurrence of a rising(+) or falling(-) edge of the binary signal with which it is labelled. Black transitions are used for input signals, white ones for output signals. STGs express causal dependence, independence and exclusion (choice, conflict) between signal edges. However, conventional STGs are unable to express certain kinds of asynchronous behaviour such as pseudo-causalities, causal linkage, biased concurrency, and race causality. These shortcomings have been overcome by the introduction of the generalized STG (gSTG) [5]. CASCADE supports the full modelling power of the gSTG. The additional net elements needed, such as unlabelled and tc-labelled read and inhibitor arcs, have been incorporated into the Petri net editor PED [6] that outputs net data for further processing by CASCADE. This meets our first demand. To meet the second demand (choice of design style), the net data can be preprocessed and handed over to existing synthesis tools (Fig. 1). CASCADE supports speed-independent (SI) synthesis with petrify [3] and extended-burst-mode (XBM) synthesis with 3D. Support of hazard-tolerant synthesis [7] is currently being incorporated (shaded in fig 1).

STG data can be directly handed over to petrify (which includes a feasibility checker) using a format converter. Interfacing to 3D is not possible directly because 3D starts from an XBM machine (XBMM), an FSM-like specification which, if it exists, guarantees implementability [2]. However, CASCADE can derive a primitive flow table (PFT) from the gSTG, check it for XBM feasibility, and, if positive, transform it into an XBMM. Certain forms of output concurrency, not implementable by a single XBMM [2], are treated by a parallel decomposition algorithm for PFTs [4] as part of CASCADE's transformation procedure. Multiple-output-change (MOC) behaviour (in the sense of [1], where a single change of the input state causes a sequence of output-state changes), which is forbidden in a single XBMM, but may realized by a set of interacting 3D circuits, can also be handled using CASCADE, as

shown in Sect. 3.1. This enables designers to implement systematically a larger class of behaviour than ever berfore.

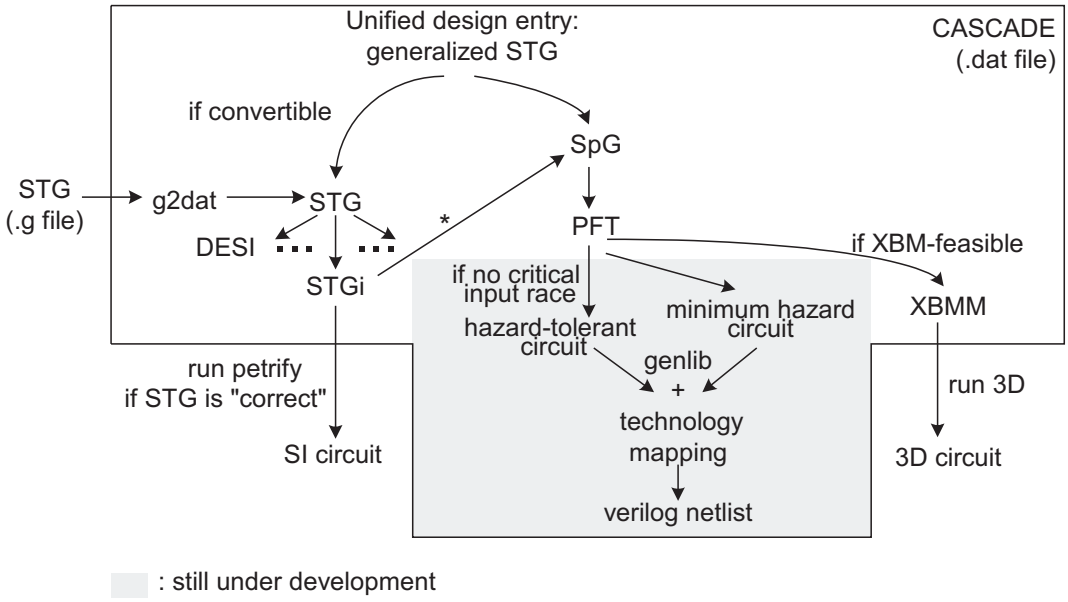

Figure 1: Comprehensive Design Method

Strengths and Weaknesses

As a standalone tool, CASCADE is very useful in synthesizing asynchronous controllers with STG as specification. Users can derive either minimum hazard equations or XBMMs from STG. CASCADE has been used internally until now. We would be very grateful for any feedback that can improve this tool.

Application domain

High-speed asynchronous controllers.

Use of Existing HDLs

The initial STG can be read from g-files (petrify). The Netlist can be generated in Verilog (early developement ...). Gate libraries are read in genlib format (from SIS).

Extent of Automation

The graphical user interface provides a 'single button sythesis', as well as access to all synthesis parameters.

Category

Synthesis - gSTG/Petri net.

Design Flow and Commercial EDA Tool Requirement

CASCADE is a standalone synthesis tool. Within CASCADE the graphical Petri net editor “PED’”is used to create a (g)STG which specifies the system's behavior. CASCADE will

derive either a single two level AND-OR-circuit, or a combination of two level circuits. If a gate library is provided, the circuits will be transformed into netlists.

Test Strategy

We are still working on netlist generation that will enable simulation of the circuit with other tools.

Current Status of Tool

The minimum hazard and XBM synthesis branch have been used internally for quite a time and are judged to be reliable. The hazard-tolerating synthesis is hardly tested and still being worked on.

Current Activities

We are currently testing the hazard-tolerating synthesis and also the technology mapping part of the tool. We have done some improvements to CASCADE's graphical user interface. The STG decomposer tool (DESI) is also being improved. Interfacing with HDL is under development.

Maintainer

Karsten Laux: *laux@eit.uni-kl.de*

Benedictus Kangsah: *kangsah@rhrk.uni-kl.de*

Tool Availability

Available from *http://www.eit.uni-kl.de/beister/eng/projects/download.html*

Future plans

We will continue to improve CASCADE as described above in current activities.

Significant Demonstrators

CASCADE has been tested with some controller specification, e.g. Fifo, VMEbus Interrupt, etc. All of the examples have been included in the CASCADE package in PED format.

References

[1] Unger, S.H. “*Asynchronous Sequential Switching Circuits*”. R.E. Krieger, reprint 1983 (original edition 1969).

[2] Yun, K.Y. “*Synthesis of Asynchronous Controllers for Heterogeneous Systems*”. PhD thesis, Stanford University (1994).

[3] Cortadella, J. “*Petrify: A tutorial for the designer of asynchronous circuits*”. Available as part of the petrify tool package from: *http://www.lsi.upc.es/jordic/ petrify*.

[4] Beister, J., Eckstein, G., Wollowski, R.. “*From STG to Extended-Burst-Mode Machines*”. In: Proc. of the 5th Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, Barcelona (April 1999). IEEE Computer Society Press.

[5] Wollowski, R., Beister, J. “*Comprehensive Causal Specification of Asynchronous Controller and Arbiter Behaviour*”. In: Yakovlev, A., Gomes, L., Lavagno, L. (eds.): Hardware Design and Petri Nets. Kluwer Academic Publishers, Boston (2000) 3-32.

[6] Tiedemann, R.: Dokumentation PED Version 4.3 (Benutzerleitfaden). Technical Report, Cottbus Technical University (June 1997).

[7] Eckstein, G. "*Logischer Entwurf hasardtoleranter asynchroner Schaltwerksverbünde (Logical design of hazard-tolerant communicating asynchronous circuits)*". Doctoral dissertation, University of Kaiserslautern. Fortschritt-Berichte VDI, Reihe 20, Nr.324. VDI-Verlag, Düsseldorf (2000).

[8] Kondratyev, A., Kishinevsky, M., Yakovlev, A. "*Hazard-free implementation of speed-independent circuits"*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 17 (September 1998).

Tool/Methodology: CAST (Caltech Asynchronous Synthesis Tools)

Developers: Alain J. Martin, Mika Nyström

Organisation: California Institute of Technology

General Summary

CAST is the name of the suite of design tools developed to support the Caltech synthesis method for asynchronous QDI (*quasi delay-insensitive*) circuits. The method is based on synthesis by program transformations. The system to be designed is first described in the high-level language CHP. This description is refined, manually or automatically, into a more concurrent version. Once the desired degree of concurrency among the CHP components and the desired granularity for each CHP process are achieved, the CHP processes are translated into the HSE (*handshaking expansion*) notation, in which all communications are replaced with handshake protocols, and all variables are implemented in terms of booleans. The HSE representation can also be modified. HSE processes can be decomposed further, but the most important transformation at this level is what is called “reshuffling”: handshake protocols on different channels are made to overlap to some degree in order to improve efficiency or simplify the implementation. Finally, all explicit sequencing is removed by translating the HSE representation into a PRS (*production-rule set*). The PRS representation contains almost the same information as a traditional SPICE netlist, and it is therefore considered the target of the logic synthesis. The CAST toolset consists of:

- high-level synthesis tools including both source-to-source (CHP) decomposition tools and logic-synthesis tool. (2)

- simulation tools at various levels of representation - both logic and performance simulation including energy and timing, and electrical simulation, and tools for cosimulation of a design mixing different levels of representation

- low-level tools for translation of production-rule sets into layout, as well as placement and routing, and tools for checking electrical properties of layout (charge sharing, slew rate, etc).

The CAST toolset provides several alternative paths for designers: the standard CAST solution (described in more detail below and shown in the figure) is a specialization of the general method targeting a specific building block called PCHB. Other paths are also possible starting from any HSE representation.

The different programs of the suite are structured so as to make it possible for the designers to tailor the tools to their styles, needs, and experiences, by carefully designing the interfaces in such a way that replacing one tool with another should be easy. The current design procedure embodied in the CAST tools represent a specialization of the asynchronous design techniques pioneered at Caltech in the 1980s and used to design the world's first asynchronous microprocessor rather than a departure from those techniques. The traditional technique consisted of a series of stepwise refinements starting at the CHP level and ending in finished PRS, proceeding through the HSE level of description. The HSE level was important in this

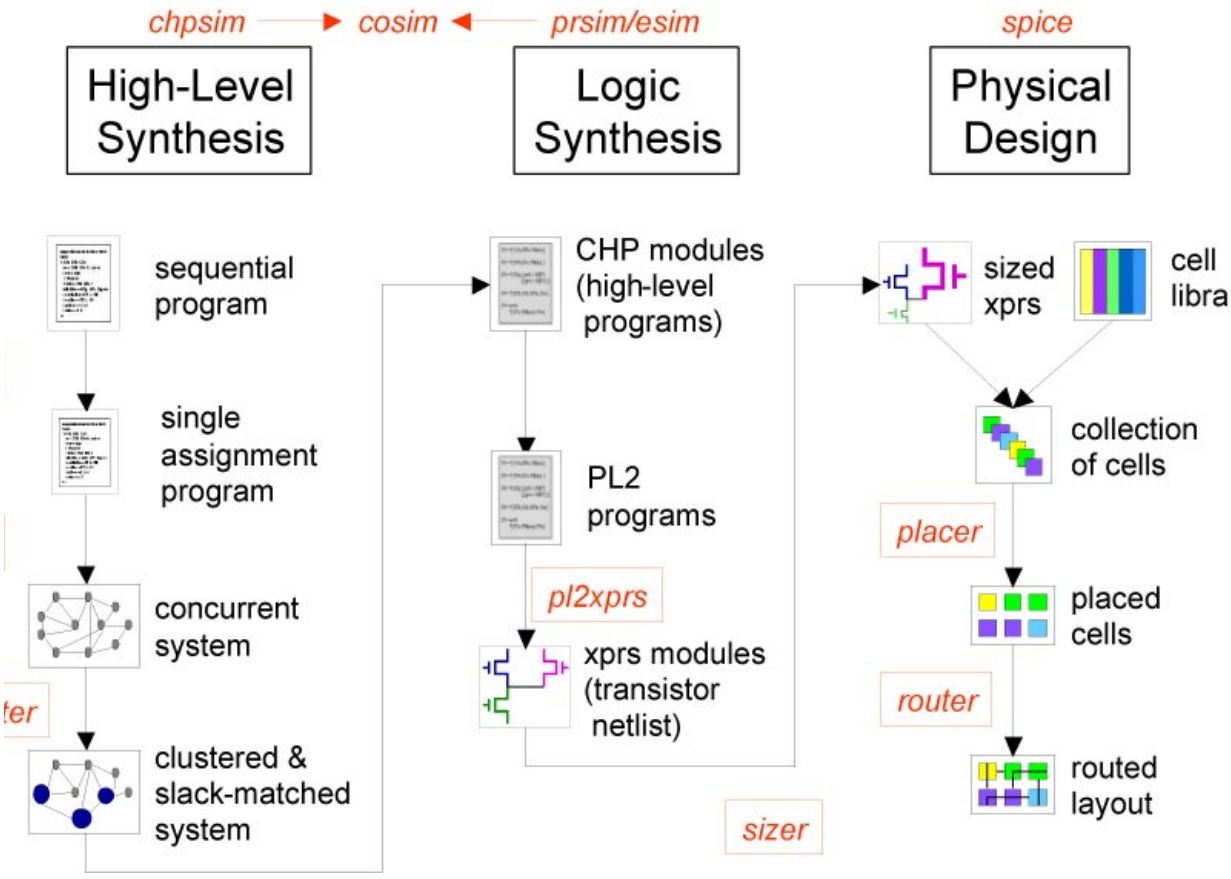

Figure 1: The CAST Toolset

design style because this is the level at which all abstract channels are replaced with booleans and all inter-process sequencing decisions (e.g., channel handshakes) are specified. The HSE level has lost of its importance in the current generation of the tools because the current generation uses very few unique HSE reshufflings, and therefore it is unnecessary to repeat the work of compiling from HSE to PRS for each and every process that is designed. Large systems, such as the MiniMIPS processor and the Lutonium microcontroller, have been designed using as few as three different HSE "templates" for the vast majority of circuits. A small fraction of the circuits still present special design problems, however (in the Lutonium, six-transistor SRAM memories were an example), and in these cases, hand design using HSE is still the norm. A tool for synthesizing those designs automatically from HSE to PRS (aptly named *hse2prs*) is included in CAST.

Strengths & Weaknesses

CAST does not yet integrate a testing procedure.

Application Domain

CAST is a general tool suite with emphasis on high-performance microprocessor design.

Use of Existing HDLs

For high-level description, the CAST tools use the language CHP, which was developed and refined at Caltech over the last decade, rather than VHDL or Verilog. However, an alternative is provided in the form of a subset of VHDL that exactly implements the constructs of CHP. This language is called CHDL. A user designing in CHDL will be in the familiar environment of VHDL. The resulting code is translated into CHP at little cost. From then on, the design flow is the same.

Extent of Automation

Our goal is to make the CAST tools entirely automatic, from high-level representation to the final layout. Currently, there are only some small gaps in the tool flow that are in the process of being filled in. More important than the extent of the automation, however, is the way in which the automation has been achieved. The CAST tools are designed in accordance with the idea of "designer-assisted compilation": the idea is that the designer can affect any step of the compilation procedure and insert his own design. This goes for all levels from the highest decompositions to the lowest-level layout descriptions. The CAST tools thus make it easy to mix high-level automatically decomposed circuits with hand-designed SRAM cells and other special circuits. All the different levels of description are presented in one unified representation to make this easy.

Category

QDI-Synthesis

Design Flow and Commercial EDA Requirement

The CAST toolset is built around a way of designing circuits that consists of a sequence of systematic, provably correct transformations that take the designer from an initial specification to a final layout geometry. As we have seen, this sequence of transformations passes through a number of representations, e.g., CHP, HSE, PRS, transistor netlists, and various representations of layout geometries. The tools support several transformation paths, and users are free to add their own, using their own tools or standard commercial tools from other commercial CAD vendors.

High-Level Synthesis. The first step in the CAST design flow is process decomposition, which transforms sequential high-level descriptions of circuit behavior into a system of communicating modules. Each module is still expressed in a high-level language and they are normally each synthesized individually at lower levels. The goals of process decomposition are to expose concurrency and facilitate low-level synthesis while producing a system with an acceptable throughput but not a surfeit of communications. (In most QDI systems, the computation of values consumes significantly less energy than the communication of these values.) Previous approaches to automated process decomposition have been syntax-directed and unable to produce modules suited for implementation as the fine-grain pipeline stages (*precharge half-buffers*, or *PCHB*) used in the high-performance MiniMIPS and Lutonium asynchronous microprocessors. The CAST tool flow features *data-driven decomposition* (DDD), the first decomposition method to target the fast PCHB asynchronous circuit family. The CAST tool chain gives users the freedom to choose between DDD, syntax-directed decomposition, and performing the process decomposition by hand. DDD creates a working concurrent system where every module can be implemented by a PCHB circuit. DDD starts by

converting the original specification into DSA *dynamic single-assignment* form and maintaining it in that form; it then proceeds by process decomposition through *projection*.. The last stage of DDD is to cluster the DSA modules into larger modules to improve energy (reducing the number of communications in the system) and performance (cutting forward latency while still running at the desired throughput). The final output is a concurrent system where communicating modules may implement the computations of one or multiple variables, or may be simple buffers inserted to improve system performance. All modules fit the PCHB circuit template. CAST contains tools that convert deterministic CHP programs into DSA form, analyze the data dependencies in a DSA program and output a system of CHP modules equivalent to the original sequential specification. A tool that implements the final clustering stage of DDD is currently under development.

Production Rules to Layout. The CAST toolset takes the Production Rule Set (PRS) as the boundary between logical and physical design; production rules are the target of the logical design and the specification for the physical design.However, a production-rule set is a purely logical specification, and it is not sufficient for driving the physical design tools. In order to provide a specific enough (but not over-specific) description of the system, the CAST toolset uses a new representation called the Extended-Production-Rule Set (XPRS). This representation specifies transistor-gate ordering and transistor-gate widths, but it does not specify the complete circuit topology, nor does it specify any other geometry information. The XPRS notation is ideal for transistor sizing and it is also ideal for human-produced low-level descriptions: using XPRS, a designer can specify all relevant details about an asynchronous circuit implementation without having to edit actual chip layout directly. The introduction of XPRS subdivides what was formerly one task (sizing, gate ordering, and specification of the netlist) into two tasks (sizing and gate ordering on the one hand and specification of the netlist on the other). The CAST tools convert the standard PRS into XPRS as the first step in the physical-design flow. This conversion is done in one of three ways: *i*) Automatically through gate matching. (*ii*) Automatically through XPRS generation from PRS. (*iii*) Manually or by logical-design tools. The three ways are used as follows: the automatic methods are used when a PRS is given. First of all, gate matching is performed: the CAST system is able to match a given PRS against a gate library whose cells are described in XPRS – in this case, the presence of a cell in the gate library is taken to mean that the transistor-gate ordering is arbitrary, and logically equivalent cells are matched against the given PRS. Secondly, remaining PRs are converted by a special XPRS generator: this generator makes the decisions regarding transistor-gate ordering and gate sharing; this is the least preferred approach because the XPRS generator has to be conservative about its designs in order to guarantee that they function properly. The final method of generating XPRS is the simplest: the user simply specifies the gate ordering. Normally, however, the “user” is a higher-level tool in the CAST logical-design suite; this tool will have the necessary information to pick a reasonable gate ordering and sharing. In the current system, automatic gate matching has not yet been implemented; this is not a major drawback because most circuits are compiler-generated anyhow, and the compiler is aware of the structure of the standard-cell library.

Placement and Routing. The CAST toolchain is extremely flexible with regard to cell placement. Cells can either be placed manually by the designer by leaving the appropriate directives in the CAST code, or the placement can be done automatically. If it is done automatically, special directives can still be used in order to perform datapath placement – the regularity of a datapath means that extra information is available in order to optimize the

routes. The CAST03 system makes it easy for the designer to specify this extra information. Routing is performed by a proprietary CAST router. The CAST router routes nets one at a time, it supports rip-up-and-reroute for batch mode “hands-off” routing, and it works with all standard ASIC processes. The router makes it very easy to combine standard cells with hand-drawn layout, in keeping with the CAST philosophy of designer-assisted compilation.

Commercial EDA Requirement. Currently, the CAST tools do not require the use of any commercial tools. It is our intention to keep the use of commercial tools as small and as optional as possible in order to enable the entire system to be used for a very small start-up cost.

Test Strategy

Currently, CAST does not have any special features devoted to testing.

Current Status

The status of the CAST tools project can be summarized as follows. At the logic-synthesis level, an automatic procedure exists for decomposing any given CHP or CHDL program into a network of small components. Also, the core of the logic synthesis – the transformation from CHP to PRS – has been formalized. At the simulation level, the framework of the CAST toolset has been defined and the interfaces have been delineated. A CHP simulator now exists, and a new method for mixed-level simulation or cosimulation has been defined. At the physical-design level, a standard-cell library has been defined and built. The front-end of a cell generator (*stackgen*) has been developed, as well as two placers (one constraint-based and one based on simulated annealing) and a router. An extended version of the PRS language, called XPRS, has been defined that contains information about transistor ordering and sizing. XPRS makes it possible to layout a chip without need to edit the layout manually.

Current Activities

Maintainer

The CAST tools are maintained by the Asynchronous VLSI Group of the Computer Science Department at the California Institute of Technology, Pasadena, California, U.S.A., and by Situs Logic, Pasadena, California, U.S.A.

Tool Availability

The CAST tools are currently only available internally at Caltech and Situs Logic, although earlier versions have slightly wider circulation. Situs Logic is in the process of commercializing the tools for a wider range of users.

Future Plans

N/A

Significant demonstrators

Previous versions of the CAST tools have been used to design the Caltech Asynchronous Microprocessor (the world's first asynchronous microprocessor), the MiniMIPS processor (a two-million-transistor quasi delay-insensitive clone of the MIPS R3000), and various other chips. The current version of the CAST tools is being used to design the Lutonium microcontroller, an Intel-8051 compatible microcontroller.

Tool/Methodology: CCS-based specification

Developer: Graham Birtwistle

Organisation: School of Computing, Leeds University, UK

Summary

The work of the Leeds based group is methodological not tool building. We are interested in finding ways to specify the control signals in large realistic circuits. We specify in the CCS notation and use its CONCURRENCY WORKBENCH support tool to minimise, equivalence and property check.

The Leeds work was preceded by work at Calgary, Canada, where we used CCS to specify cell libraries and then design three variations of a (very) small microprocessor (4-phase (RTZ), 4-phase with exception handing, and 2-phase pipelined). The last machine was laid out by Tom Borsodi from our CCS specification in Actel FPGA technology using Erik Brunvand's thesis as guideline. One chip was required for the 8-deep register bank; the rest fitted onto another.

With Ying Liu (also at Calgary), we worked with the Amulet group at Manchester University formalising the design of AMULET1, their 2-phase asynchronous version of ARM6. We put a lot of work into picking an appropriate notation for extracting the specification from the Manchester architects in a manner that made them feel comfortable and from which we could derive the CCS specifications mechanically. We also put a lot of time into coming up with the right levels of abstraction. We formalised and checked the 5 main floor plan elements. We also have an instruction level specification which clarifies the links amongst the floorplan elements. We are now looking at facts and figures on 2- and 4-phase pipelines and working round the problems encountered with Ying. The latest (Harvard-style) architecture is temporarily called TK (short for trinket - a cheap amulet, according to Websters). Once this is sorted, then a 4-phase version will be studied.

Strengths and Weaknesses

CCS is a small language with a fully defined syntax and semantics; with well defined equivalence rules; and a match with the powerful modal mu calculus for property checking. CCS is a suitable notation for specifying and reasoning about control signals in 2-phase or 4-phase asynchronous systems. It has been used to reason about control in circuits from the gate level and above. Its support tool, the CWB, is public domain. It has been found to be robust and reliable and has been used to minimise and property check pipelined circuits with 10^{50} or more states.

Its weaknesses include:

- it does not handle data values well being best suited to the study of control signals only;
- it is however not a programming language and CCS descriptions cannot be run as simulations;
- its notion of time is "before" rather than numerical so it cannot be used for performance estimations.

Application Domain

Modelling of asynchronous 2- and 4-phase pipelines and proving theorems about their state spaces; the specification and property checking of AMULET like (but simpler) microprocessor designs.

Use of existing HDLs

CCS is not a HDL

Extent of Automation

CCS specifications are entered into the CWB and syntactically checked. They may then be minimised to the least equivalent state machine; property checked (deadlock etc.) and checked for equivalence against other definitions (e.g. an implementation) by built-in procedures.

Categories

Verifier

Design Flow and Commercial EDA Tool Requirement

Stand alone.

Test Strategy

Property checking and equivalence checking.

Current Status of Tool

See Concurrency Workbench, Edinburgh University.

Current Activities

- proving facts about 2- and 4-phase pipelines.
- extending a basic RTL model to include exception handling, register forwarding, etc.

Maintainer

Professor Graham Birtwhistle has now retired from his post at the University of Leeds.

Availability

CCS was designed by Robin Milner while at Edinburgh University. Its public domain CWB support tool is freely available from CS at Edinburgh.

Future Plans

Continuation of the current program

Demonstrators

Published papers and theses available via: *www.comp.leeds.ac.uk/research/asynch/asynch.html*

Tool/Methodology: Clp

Victor Khomenko

School of Computing Science, University of Newcastle upon Tyne

Summary

Clp-- a Model Checking Engine Based on Petri Net Unfoldings is an integer programming based model checker. It can formally verify various safety properties (e.g. deadlock-freeness and mutex), and detect of coding (CSC and USC) conflicts in STGs. Clp employs finite complete prefixes of Petri net unfoldings (e.g. those produced by the Punf tool [3].

Strengths and Weaknesses

Memory efficient and quite fast, though the performance on large benchmarks might be not entirely satisfactory; we deal with this problem in the VerySAT tool (see current status below).

Application domain

verification and synthesis of self-timed circuits.

Use of existing HDLs

none (works on finite prefixes in the `.mci' format).

Extent of automation

fully automatic.

Category

synthesis=>STG/Petri net

Design flow

Clp is intended as a powerful model checking engine to be used by other applications. Currently, prefixes produced by Punf [3] can be used by Clp for detection of coding (CSC and USC) conflicts and normalcy violations. Clp, together with Punf and ConfRes tools, comprise a framework for detection and resolution of coding conflicts in STGs [6,7].

Test strategy

Clp can formally verify many safety properties (e.g. deadlock-freeness and mutex).

Current status of the Tool

Clp is fully operational. It is integrated into the PEP tool [1]. We work on a tool VerySAT offering similar functionality but employing a SAT solver rather than a specialized integer-programming one, which is often faster [8].

Maintainer

Victor Khomenko (Victor.Khomenko@ncl.ac.uk).

Availability

Available for research purposes from

http://www.cs.ncl.ac.uk/people/victor.khomenko/home.formal/tools/tools.html

Future plans

We plan to create a full design cycle for self-timed circuits based on Petri net unfoldings and not involving building the state space at any stage. VerySAT is to replace Clp in near future as its performance is much better.

References

[1] E.Best and B.Grahlmann: *"PEP: Documentation and User Guide, Version 1.4. Manual"* (1995).

[2] V.Khomenko: "*Model Checking Based on Prefixes of Petri Net Unfoldings"*. PhD Thesis, Department of Computing Science, University of Newcastle (2002).

[3] V.Khomenko: *"Punf: Documentation and User Guide"*, Version 6.01. Manual (2002).

[4] V.Khomenko: "*Clp: Documentation and User Guide*". Version 3.01beta. Manual (2002).

[5] V.Khomenko and M.Koutny: "*LP Deadlock Checking Using Partial Order Dependencies"*. CONCUR'2000, LNCS 1877 (2000) 410-425.

[6] V.Khomenko, M.Koutny, and A.Yakovlev: "*Detecting State Coding Conflicts in STGs Using Integer Programming*". DATE'2002, IEEE Comp. Soc. Press (2002) 338-345.

[7] A.Madalinski, A.Bystrov, V.Khomenko, and A.Yakovlev: "*Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design"*. DATE'2003, IEEE Comp. Soc. Press (2003) to appear.

[8] V.Khomenko, M.Koutny, and A.Yakovlev: "*Detecting State Coding Conflicts in STG Unfoldings Using SAT*. ICACSD'2003", IEEE Comp. Soc. Press (2003) submitted paper.

Tool/Methodology: ConfRes

Developer: A.Madalinski

Organisation: University of Newcastle upon Tyne, UK

Summary

The tool supports semi-automated resolution of Complete State Coding (CSC) conflicts in asynchronous circuit specification given as Signal Transition Graphs (STGs) and display them as partial orders (finite and complete prefixes of STG unfoldings). Being more efficient than the automated methods the manual approach requires a significant effort from the designer. The tool ConfRes assists the designer by visualising the conflicts cores, their superposition and the constraints on signal insertion.

Cores extend the known concept of complementary sets. Only those complementary sets are used which are not combinations of others . The advantage of using cores is that only those parts of STGs are considered, which cause coding conflicts, rather than the complete set of conflicts. Since the number of cores is usually much smaller than the number of coding conflicts, this approach saves the designer from analysing large amounts of information. Moreover, cores are represented at the level of the STG unfolding prefix, which is a convenient model for understanding the behaviour of the system due to its simple branching structure and acyclicity.

Cores are important for resolving coding conflicts. They can be eliminatedby adding auxiliary signals and by concurrency reduction, respectively.The former introduces additional internal signals to disambiguate encoding conflicts and the latter reduces the state space in the STG's reachability graph and thus potential encoding conflicts. The resolution process uses the partial order model and employs the visualisation concept as follows: (a) shows the superposition of cores by meansof a 'height map' (b) only those cores are displayed which are relevant to selected part of the specification, i.e. cores which are extracted from the height map and (c) the constraints on insertion identify a small part of the specification rather than the entire design. In addition, heuristics are used to pre-compute solutions, which are suggested to the designer. These can be used as guidelines, however, the designer is free to intervene at any stage and choose an alternative solution in order to account the design constraints.

ConfRes takes an STG in the '.g' format supported by Petrify, an STG-based synthesis tool. It uses Punf, a Petri net unfolder, to produce a finite and complete prefix of the STG, and either Clp[2], a linear programming model checker or VerySAT [4], a SAT based model checker, to detect coding conflicts in the STG. Both of these tools are described elsewhere in this report. After the detection of conflicts, cores are computed and the resolution process is applied. The tool guides the designer through all the stages. During this process the cores and the corresponding height map are visualised using Dot [5], a graph drawing software by AT&T , and the designer can interactively insert new signals to obtain a customised solution.

Strengths and Weaknesses

Manual approach requires human participation. For this reason the designer should be familiar with STG-based design. The tool, however, minimises the efforts of the designer by using a

compact model, the concept of cores and constraints on insertion. In addition, it suggests solutions, which can be used as a guideline.

Application Domain

High-speed asynchronous controllers, e.g. interface logic, pipeline controllers

Use of Existing HDLs

none

Extent of Automation

The tool is aimed at facilitating a manual refinement of an STG with coding conflicts. Its guides the designer through the steps of the resolution process, where the designer is free to choose a location for the signal insertion.

Category

Synthesis STG/Petri net

Design Flow and Commercial EDA Tool Requirement

Any STG-based synthesis, where next-state functions are computed, needs solving coding conflicts. In particular, ConfRes can be employed in combination with STG unfolding based model checker and synthesis tool VerySAT [4] to solve the CSC problem. Alternatively, ConfRes can be used in combination with the state-based synthesis tool Petrify [1] to enforce the CSC condition, and thus to obtain a tailor made solution.

Test Strategy

N/A

Current Status of Tool

ConfRes is at the alpha-stage

Current activities

Extensive testing

Maintainer

A. Madalinski (a.a.madalinski@ncl.ac.uk)

Tool Availability

http://async.org.uk/movie/

Future Plans

Greater extension of automation.

Demonstrators

none

References

[1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev: "*Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers*". In XI Conference on Design of Integrated Circuits and Systems, (1996).

[2] V. Khomenko: "*Clp Documentation and User Guide*". Department of Computing Science, University of Newcastle (2002).

[3] V. Khomenko: "*Punf Documentation and User Guide*". Department of Computing Science, University of Newcastle (2002).

[4] V. Khomenko, M. Koutny, and A. Yakovlev: "*Logic synthesis avoiding state space explosion*". In Int. Conf. on Application of Concurrency to System Design. IEEE Computer Society Press, (2004 - to appear).

[5] E. Koutsofios, and S. North: "*Dot User's Manual*", AT&T Labs-Research (2002).

[6] A. Madalinski: "*ConfRes: Interactive coding conflict resolver based on core visualisation*". In Int. Conf. on Application of Concurrency to System Design. IEEE Computer Society Press, (2003).

[7] A. Madalinski, V. Khomenko, A. Bystrov, and A. Yakovlev: "*Visualisation and Resolution of Coding Conflicts in Asynchronous Circuit Design*". IEE Proceedings, Computers and Digital Techniques, Special Issue on Best Papers from DATE03, (2003).

Tool/Methodology: DESI[1] (DEcomposer SIgnal Transition Graph)

Developer: B. Kangsah, R.Wollowski, W. Vogler, and J. Beister

Organisation: Kaiserslautern University of Technology

Summary

Signal Transition Graphs (STGs) are a version of Petri nets for the specification of asynchronous circuit behaviour. As a first step in the indirect synthesis of a circuit corresponding to a given STG N, one usually constructs the reachability graph (e.g. when using the tool petrify) or the step graph (e.g. using CASCADE [2]). A serious problem - state explosion - may occur when constructing such a graph: the number r of reachable states (markings) may become too large to be handled due to insufficient storage space or too long CPU times. To avoid state explosion, one could try to decompose the STG N into components Ci (and thus, the circuit into modules). The reachability graphs of the Ci, taken together, can be much smaller than r since r might be the product of their sizes. Even if this is not achieved, decomposition can reduce design effort and save circuit area. Where N may have to be synthesized by heuristic methods, its components Ci may even be handled by exact methods yielding optimal results. Decomposition can also be useful aside from size considerations: there are examples where N cannot be handled by a certain synthesis method, while its Ci can (e.g. deriving a set of XBM machines from an STG [1]). It may also be possible to extract library elements; this is particularly valuable for arbiters, which are difficult to design.

We have presented a decomposition algorithm [5] that is based on that of Chu [3] but is much more generally applicable. In particular, there is no restriction to live and safe free-choice nets or to marked graphs, and labels are not required to occur only once. A formal proof based on a formal correctness criterion has been given. The algorithm starts with a given partition of the set of output variables: each Ci is responsible for one block of the partition. The Cis are then extracted from the STG by transition contraction, care being taken to keep only the relevant input signals, which may be global inputs or outputs of other components. The kernel of the algorithm has been implemented in DESI (DEcomposer SIgnal Transition Graph). DESI is originally design as part of CASCADE, which can also forward results to other synthesis tools such as petrify and 3D.

Strengths & Weaknesses

With STG decomposition one may be able to:

- avoid state explosion
- reduce design effort
- save circuit area
- derive a set of XBM machines from an STG

1. This work was partially supported by the DFG-project ` STG-Dekomposition' Vo615/7-1 / Wo814/1-1.

- extract library elements.

The STGs to be decomposed has to fulfill the following requirements:

- no internal transitions
- no auto-concurrency
- no structural auto-conflicts
- no i/o conflicts.

For more detail, please refer to [5].

In the current version of DESI one should choose the output partitions manually. The blocks of the partitions must be written in the DESI configuration file. This will be improved in the future.

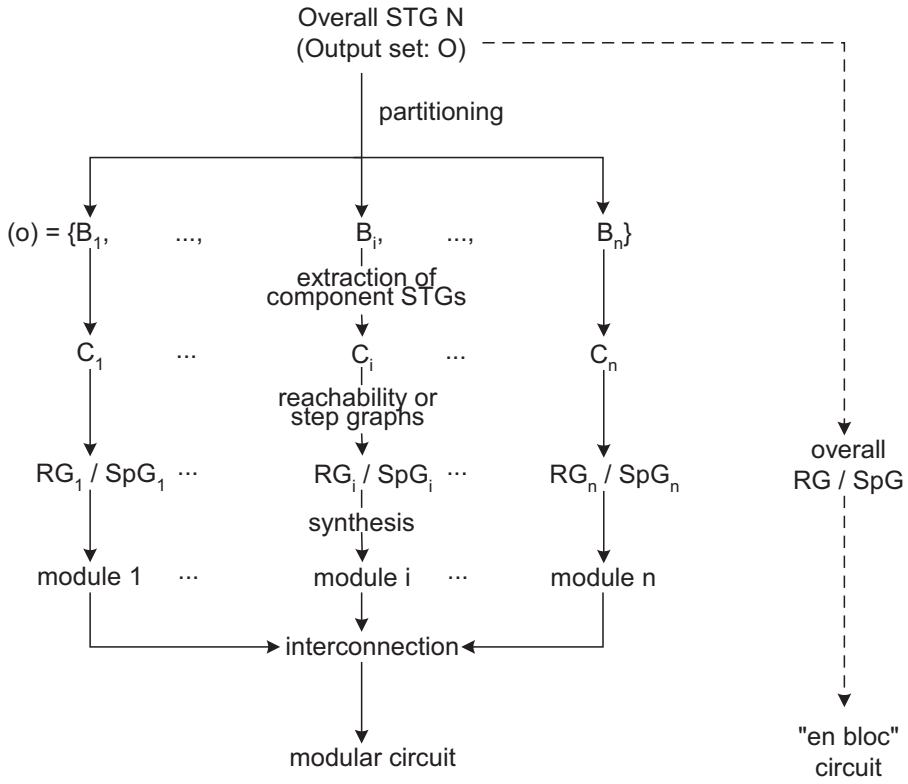

Figure 1: DESI Tool Flow

Application Domain

Complex STG specifications for synthesis of asynchronous circuits.

Use of Existing HDLs

None

Extent of Automation

The whole algorithm has been implemented, i.e. decomposition is performed automatically. One needs only specify the partition and the output file type (dat-file for CASCADE, g-file for petrify, dot-file for graphical view)

Category

Specification with STG

Design Flow and Commercial EDA Requirement

We propose a modular design by applying STG decomposition:

- decompose the STG N into components Ci
- for each Ci synthesize a module
- compose all modules and reach a modular circuit

DESI is designed to be used with academic tools like CASCADE, petrify and 3D. If you are interested to connect your tool with DESI, we would be happy to help you create the interface between DESI and yours.

Test Strategy

Correctness is proven !!! [5]

Current Status of Tool

Current Activities

DESI is continously being improved, currently by fine tuning and optimization.

Maintainer

Benedictus Kangsah : *kangsah@rhrk.uni-kl.de*

Tool Availability

DESI is free for academic use. We would be very grateful for any feedback or comment. You can download DESI from our website:

http://www.eit.uni-kl.de/beister/eng/projects/download.html

Future Plans

We will try to relax some of the STG requirements (see strengths and weaknesses). We also want to study quality criteria for decompositions and methods for finding good decompositions.

Significant demonstrators

You can view several examples of step by step decomposition at our website:

http://www.eit.uni-kl.de/beister/eng/projects/deco_examples/main_examples.html

You can also observe this step by step decomposition by setting the corresponding output parameter in the DESI configuration file (see user guide in DESI distribution package). In [4] you can find DESI result table of some examples.

References

[1] J. Beister, G. Eckstein, and R. Wollowski. "*From STG to Extended-Burst-Mode Machines*". Proc. 5th International Symposium on Advanced Research in Asynchronous Circuits and Systems. IEEE Computer Society Press, 1999.

[2] J. Beister, G. Eckstein, and R. Wollowski. "*Cascade: a tool kernel supporting a comprehensive design method for asynchronous controllers*". In M. Nielsen, editor, Applications and Theory of Petri Nets 2000, Lect. Notes Comp. Sci. 1825, 445-454. Springer, 2000.

[3] T.-A. Chu. "*Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications*". PhD thesis, MIT, 1987.

[4] B. Kangsah, R. Wollowski, W. Vogler, J. Beister. "*DESI: a Tool for Decomposing Signal Transition Graphs*". Presentation in 3rd ACiD-WG Workshop, Crete, 2003.

[5] W. Vogler, and R. Wollowski. "*Decomposition in Asynchronous Circuit Design*". In J. Cortadella, A. Yakovlev, G. Rozenberg, editor, Concurrency and Hardware Design, Lect. Notes Comp. Sci. 2549, 152-190. Springer, 2002.

This work was partially supported by the DFG-project 'STG-Dekomposition' Vo615/7-1 / Wo814/1-1.

Tool/Methodology: di2pn, syndi and diana

Developers: Dennis Furey, Mark Josephs and Hemangee Kapoor

Organisation: London South Bank University

Summary

A variety of notations for the specification of delay-insensitive modules (such as the handshake components that are targeted by the Tangram and Balsa silicon compilers) have been proposed. The input format for the tool *di2pn* is one such notation, a structured programming language in which input/output bursts serve as primitive statements.

The language (Delay-Insensitive Sequential Processes), a variant of Hoare's Communicating Sequential Processes and of Josephs and Udding's DI-Algebra, is aimed at designers of asynchronous circuits and systems. When modules are to be implemented in digital logic, inputs and outputs are usually interpreted as signal transitions.

di2pn automatically translates programs into Petri Nets, the latter being less structured, but more amenable to operational reasoning. di2pn can thus be used as a front-end to tools that operate upon Petri Nets, e.g., for the purposes of simulation, analysis and logic synthesis.

di2pn produces Petri Nets in the same text-file format as used by Petrify, an enhanced verison of the ASTG format devised for SIS. Auxiliary tools, namely, *pn2dot* and *pn2lola*, facilitate conversion to other formats.

di2pn is most closely related to Mallon's *digg* tool, which translates terms in DI-Algebra into State-Graphs rather than Petri Nets. The beta version of di2pn adopted the same input format as digg, but compatibility was abandoned as the language of Delay-Insensitive Sequential Processes developed.

As an alternative to di2pn, designers may prefer to use the newer tool, *syndi*. The input format is different from that of di2pn, supporting many of the language features that make functional programming so powerful. Besides translation to Petri nets, syndi also supports compilation to delay-insensitive circuits (DI netlists) or to virtual code executable files.

Finally, an analysis tool, *diana*, is available. This works with descriptions, either in the form of Petri nets (generated by di2pn or syndi) or in the form of DI netlists (generated by syndi). Among other things, it can be used to verify that an implementation meets its specification.

Strengths and Weaknesses

The language accepted by *di2pn* offers a concise, structured way of describing module behaviour. A user-friendly syntax has been adopted. Programs are meaningful without the designer having to understand the way *di2pn* translates them into Petri Nets.

Currently, the language only allows data communication to be expressed for delay-insensitive code words; data paths are otherwise outside the scope of *di2pn*.

Whereas it is expected that users of di2pn will rely upon Petrify for the actual synthesis of a netlist consisting of complex gates or generalised C-elements, syndi incorporates novel

strategies and algorithms for automatic decomposition and synthesis of a netlist consisting of delay-insensitive logic blocks (including arbiters).

Unlike some verification tools, diana can handle non-trivial circuits and will take into account progress properties, as well as safety properties, when performing analyses and comparisons.

Application Domain

Asynchronous controllers.

Use of Existing HDLs

None

Extent of Automation

The process specification has to be written manually, and everything else about di2pn and syndi is automated. diana takes the output of those tools and its analyses are automated.

Categories:

1. Notation to notation conversion. (di2pn and syndi)

2. Front-end (to simulation, analysis and synthesis tools). (di2pn and syndi)

3. Synthesis of netlists. (syndi)

4. Formal Verification. (diana)

Design Flow and Commercial EDA Tool Requirement

di2pn can be used to produce a “snippet” (open Petri Net) from a structured program capturing input-output behaviour.

di2pn can also be used to produce a closed Petri Net from a pair of programs (describing a module and its environment). The net can then be input into the Petrify tool for logic synthesis (or for analysis). The user can thus synthesise an asynchronous control circuit from the programs, ignoring the net.

When refining a DISP specification by hand (e.g. decomposing it into parallel components), one can use diana to check that this has been done correctly. (One must first translate the programs into Petri nets using di2pn.)

syndi can be used in place of di2pn.

pn2dot and *pn2lola* convert the format of the Petri Net produced by *di2pn* so that it can be displayed, simulated and analysed by tools that use a different input format from Petrify.

Commercial EDA tools are not required.

Test Strategy

Additional circuitry that tests for faults would have to be designed separately.

Current Status of Tools

di2pn was developed under UK Engineering & Physical Sciences Research Council project GR/M51567. The project report is available at

http://www.bcim.lsbu.ac.uk/ccsv/epsrcM51567.html.

Current activities

Experimentation with their use

Maintainer

Dennis Furey, fureyd@lsbu.ac.uk (syndi and diana; di2pn is not currently maintained)

Tool Availability

Free software with no warranty, General Public License. Download di2pn from

http://www.bcim.lsbu.ac.uk/ccsv/dac04/di2pn-0.1.1.tar.gz

and download syndi and diana from

http://myweb.lsbu.ac.uk/~fureyd/ditools/.

They require a virtual machine code interpreter available that can be downloaded from

http://myweb.lsbu.ac.uk/~fureyd/avram/.

Demonstrators

A comprehensive description of the *DISP* language and di2pn tool appears in the chapter "A programming approach to the design of logic blocks" of the book "Concurrency and Hardware Design", LNCS 2549, Springer, 2002.

di2pn has been demonstrated (e.g. at Async 2001) on several small examples, first specifying delay-insensitive modules (e.g. merges, joins, and latch controllers) as structured programs, and then using Petrify to synthesise speed-independent circuits that implement them. Slides were presented at the ACiD-WG workshop in Munich, 2002, and are available at

http://www.bcim.lsbu.ac.uk/ccsv/ACiD-WG/Workshop2FP5/Programme/JosephsSlides.pdf.

A suite of Burst-Mode benchmarks has been re-expressed in DISP and synthesised with di2pn and Petrify, *http://www.bcim.lsbu.ac.uk/ccsv/dac04/benchmarks.pdf*. As reported at DAC, June 2004, a Fork-decomposition heuristic was successfully applied to each benchmark at the DISP-level to improve the effectiveness of synthesis.

An article describing the use of di2pn with diana appears in the proceedings of ACSD, June 2004.

An article based on the beta version of di2pn was published in the proceedings of DATE 2000. Slides outlining the translation algorithm that underpins *di2pn* version 0.1.1 were presented at the ACiD-WG workshop in Grenoble, 2000, and are available at

http://tima-cmp.imag.fr/tima/cis/acid/slides/josephs.pdf.

Some of the peephole optimisations performed by di2pn before output of a Petri Net were presented at the AINT workshop in Delft, 2000.

Tool/Methodology: DGC (Digital Gate Compiler)

Developer: Oliver Kraus

Organization: University of Erlangen-Nuremberg, Germany

Summary

DGC is a synthesis tool for state machines (asynchronous and synchronous) and boolean expressions. Usually DGC generates a physical layout from the input description. Together with a description of the available gates (GENLIB format) and a description of the asynchronous or synchronous system DGC generates an EDIF or VHDL netlist. This netlist can be used to generate the layout of an integrated circuit with any EDA tool.

Parts of DGC are additionaly implemented as separate executables: BMSENCODE solves the state encoding problem, DGSOP is an exact boolean minimizer (similar to espresso) and XBM2PLA has the same functionality as 3D or MINIMALIST.

DGC supports several input formats. Most important for asynchronous circuits: DGC reads and processes extended burst mode description as introduced by 3D. Moreover DGC extends this burst mode description and allows level and edge triggered transitions for the same input signal. A scripting language allows the connection and synthesis of more than one controller.

Algorithms have been written from scratch. Among others these are state minimization, asynchronous state encoding, hazard-free minimization, hazard-free technology mapping, hazard-free technology optimization, detection of essential hazards and delay chain generation.

DGC is portable across several platforms. DGC can be installed and used on any many Unix platforms (Linux, BSD, Solaris), MAC OS X and Windows (Cygwin environment).

Strengths and Weaknesses

Strengths: Single executable that generates a netlist from a gate library and a description of one or more controllers. Implements asynchronous state and hazard-free logic optimization, hazard-free technology mapping and essential hazard elimination. Also calculates the required external delay to ensure fundamental mode operation.

Weaknesses: Long execution time for large controllers due to the use of exact algorithms. Only generates Huffman circuits.

Application Domain

Control circuits and asynchronous systems

Use of Existing HDLs

DGC can generate technology depended or independed VHDL netlists and a suitable VHDL test-bench.

Extent of Automation

Fully automated synthesis of asynchronous controller or system specification to a technology depended netlist.

Category

Synthesis, Extended Burst Mode

Design Flow and Commercial EDA Tool Requirement

The netlist produced by DGC can be used to generate a layout with usual EDA tools.

Test Strategy

DGC has an integrated formal verification tool (netlist versus boolean expressions) and an integrated gate level simulator (technology depended netlist). The tool XBM2PLA is able to produce a VHDL test bench with glitch detection.

Current Status

Development effort reduced. Bug fixing available via project home-page.

Maintainer

Oliver Kraus (olikraus@yahoo.com)

Availability

GNU GPL http://dgc.sourceforge.net

Future Plans

Implement faster algorithms.

Demonstrators

Under development at the University of Erlangen-Nuremberg.

References

[1] Kraus, Padeffke. “*Entwurfsumgebung fuer asynchrone Burst-Mode Automaten. In Methoden und Beschreibungssprachen zur Modellierung und Verikation von Schaltungen und Systemen*”, pages 86-95. Shaker Verlag, 2002. ISBN 3- 8265-9859-8.

[2] Kraus, Padeffke. “*Synthese von asynchronen "Burst-Mode" Automaten*”. In Entwurf integrierter Schaltungen und Systeme (11. EIS-Workshop), pages 39-44. VDE Verlag GmbH, 2003. ISBN 3-8007-2760-9.

[3] Kraus, Padeffke. “*XBM2PLA: A Flexible Synthesis Tool for Extended Burst Mode Machines*”. In Proc. Design, Automation and Test in Europe (DATE), pages 1092-1093, 2003.

Tool/Methodology: FIREMAPS/Process Spaces

Developers: Radu Negulescu and Robert T. Berks

Organisation: McGill University, Canada and University of Auckland, New Zealand

Summary

FIREMAPS is a tool for structured verification, design and test of asynchronous circuits and mixed synchronous/asynchronous circuits, based on concurrency theory.

Process spaces are a general theory of concurrency and provide a framework for modelling asynchronous behaviours

FIREMAPS currently supports interface design by verifying protocol compliance and by recasting an interface to a different protocol without adding circuit overheads [1]. FIREMAPS currently supports supervisory control (design equation). Burst-mode and STG-based specifications are in a prototype stage. The AND/IF standard for state-machine specifications [17] is fully supported.

Models used in formal verification:

- FIREMAPS provides direct constructors for circuit models at the gate level, switch level, cycle-accurate level, and relative timing. These constructors permit one to describe a netlist.
- Our specifications are in the process spaces formalism, which is related to CSP, Dill, etc., but are more general.
- Our timing models are metric-free (relative timing only). This is sufficient for most asynchronous circuits, and permits high efficiency and structure in verification

The techniques for manipulation and verification of asynchronous behaviors by FIREMAPS as described above are applicable early in the design stages of an asynchronous circuit. Examples include:

- When a block schematic of handshake channels is developed;
- When a cycle-level, gate-level or switch-level netlist is generated, possibly including post-synthesis peephole optimisations;
- When a library of components is developed;
- When a list of relative timing constraints is compiled for the purpose of optimal sizing of transistors and wires.

Strengths and Weaknesses

FIREMAPS permits formal verification of asynchronous circuits of high diversity (delay-insensitive, relative timing, and even multiple-clock synchronous) at diverse levels of abstraction (switch-level, gate-level, cycle-accurate level, system level, and combinations of these levels). Some of these applications (switch-level, communication refinement, etc.) are not currently supported in other asynchronous analysis methods (CSP, Dill, etc.).

FIREMAPS supports extended hierarchical verification (which can include overlapping specifications), design of discrete-state control (supervisory control / design equation), and test pattern generation for faults in asynchronous circuits.

Process spaces and FIREMAPS include new techniques for formal verification of communication refinement, peephole optimisations in synchronous and asynchronous circuits, and relative timing constraints before layout extraction and back-annotation.

Our methods provide unique techniques for the design of asynchronous interfaces, such as interface recasting whereby the interface protocol is changed without using converter circuits or other overheads [1]) and verification of compliance to asynchronous protocols.

Process spaces are related to most other asynchronous formalisms, but have a more extensive and cleaner treatment of algebraic properties, and stronger support for structured design and verification. For example, the absence of connectivity restrictions has so far permitted us to derive verification techniques for analysis of switch-level circuits, relative timing, and communication refinement, substantially exceeding the present scope of other concurrency theories for asynchronous circuits. We expect more such techniques can be derived.

FIREMAPS currently lacks integration with mainstream CAD environments (such as Cadence) and other asynchronous tools, such as hazard-free logic minimization, Petrify, and Balsa. The efficiency of FIREMAPS is not top-of-the-line among BDD-based tools, such as Petrify. Still, empirical evidence generally indicates that BDD-based tools, including FIREMAPS, are more efficient than non-BDD tools, such as Verdect and Digg, by several orders of magnitude in typical applications.

Application Domain

Applications have been demonstrated for high-speed asynchronous circuits (e.g. pulse-mode and GasP circuits), low-power asynchronous circuits (e.g. Peeters' single-rail handshake circuits, Nielsen's self-timed memory), and several other synchronous and asynchronous designs.

Use of Existing HDLs

None

Extent of Automation

Verification and diagnosis of asynchronous behaviours, peephole optimisations, communication refinement, and relative timing are implemented efficiently. - Supervisory control, interface recasting, protocol compliance, test pattern generation are implemented as prototypes and need upgrade for higher speed. - All process space operations are fully automated in FIREMAPS.

Categories

Synthesis

Design Flow and Commercial EDA Tool Requirement

The techniques for manipulation and verification of asynchronous behaviours by FIREMAPS as described above are applicable early in the design stages of an asynchronous circuit. Examples include:

- when a block schematic of handshake channels is developed;
- when a cycle-level, gate-level or switch-level netlist is generated, possibly including post-synthesis peephole optimisations
- when a library of components is developed;
- when a list of relative timing constraints is compiled for the purpose of optimal sizing of transistors and wires.

Test Strategy

We are currently developing FIREMAPS to derive test patterns and to manipulate (collapse) fault models according to the DUDES method of [15], which is extended in our approach to include arbitrating circuits and non-deterministic specifications. This method is in a prototyping stage.

Current Status of Tool

Current activities

FIREMAPS and process spaces are currently used in developing and verifying designs developed in the group led by Prof. Radu Negulescu in the Electrical and Computer Engineering Department, McGill University.

Maintainer

The contact persons for FIREMAPS maintenance and licensing are Radu Negulescu at McGill University (radu@macs.ece.mcgill.ca) or Robert Berks at University of Auckland (r.berks@cs.auckland.ac.nz). The contributions of Xiaohua Kong, Larry Weidong Ying and Mark De Clercq in developing related methods and techniques are gratefully acknowledged.

Availability

FIREMAPS can be licensed to industry that contributes funds or other forms of collaboration to the research programme of the group, and to research collaborators from academia. The average contribution from industry is approximately 10000$, and the typical licensing arrangement is a site license with unlimited use. The underlying BDD library has to be obtained independently from Carnegie-Mellon University, and it is free for academia.

Future Plans

Parts of FIREMAPS will eventually be released in the public domain, but no specific dates or plans are set at this time for such release.

Significant Demonstrators

Below we indicate several references where FIREMAPS and process spaces are presented. References [13] and [14] are currently under review, and are available only as technical reports upon request from the authors. Published references are collected at

www.macs.ece.mcgill.ca/~radu/frames/publications.html.

FIREMAPS can be tried on-line at

http://www.macs.ece.mcgill.ca/cgi-bin/cgiwrap/fm/demo_in.cgi.

The user interface of FIREMAPS and a sample case study are described in [16]. A comprehensive reference for process spaces and FIREMAPS is [7], including a study of enhanced properties of concurrent systems. Parts of the process space theory have been published in [5] and [13]. Some aspects of implementing FIREMAPS have been described in [6]; the relevant sections of [6], written by R. Negulescu, have been included in [7]. Support for interface design (interface recasting and protocol compliance) is presented in [1]; this reference treats not just asynchronous protocols, but also asynchronous models for edge-triggered and master-slave specifications. Verification of communication refinement is presented in [11], and includes mixed synchronous and asynchronous circuits. A technique for formal verification of peephole optimisations is demonstrated in [12]; this technique takes into consideration timing constraints as optimization assumptions and permits changes in the interfaces of the locally epitomized modules. A technique for efficient formal verification of mixed delay-insensitive and speed-dependent circuits is presented in [2]. Metric-free verification of relative timing constraints was introduced in [10], then applied in [8] and adopted in other asynchronous synthesis and verification methods as well. This approach permits to model constraints of the form that one path in a circuit always takes longer than another path. Metric-free verification is orders of magnitude more efficient than approaches based on numeric values of delays, and, in addition, it permits verification before layout, when wire delays are not yet known. Mixed switch-level, gate-level, and system-level verification for asynchronous circuits has been demonstrated in [9] and [3]. Support for discrete-state control and some applications thereof have been demonstrated in [4] and [14]

References:

[1] R. Negulescu, X. Kong. "*Semi-hiding operators and the analysis of active-edge specifications for digital circuits*". In Proceedings of the International Conference on Application of Concurrency to System Design (ICACSD), 2001.

[2] R. Berks, R. Negulescu. "*Partial-order correctness-preserving properties of delay-insensitive circuits*". In Proceedings of the Seventh International Symposium for Advanced Research in Asynchronous Circuits and Systems (ASYNC), 2001. (Best paper award finalist.)

[3] X.Kong and R. Negulescu. "*Verification of pulse-mode asynchronous circuits*". In Asia and South Pacific Design Automation Conference (ASP-DAC), 2001.

[4] H. Hallal, R. Negulescu, and A. Petrenko. "*Design of divergence-free protocol converters using supervisory control techniques*". In Proceedings of the Seventh IEEE International Conference on Electronics, Circuits and Systems, 2000.

[5] R. Negulescu. "*Process spaces*". In Proceedings of the Eleventh International Conference on Concurrency Theory (CONCUR), 2000.

[6] J.A. Brzozowski, R. Negulescu. "*Automata of asynchronous behaviors*". Theoretical Computer Science 231(1):113-128, 2000.

[7] R. Negulescu. "*Process Spaces and Formal Verification of Asynchronous Circuits*". PhD Thesis, Department of Computer Science, University of Waterloo, Canada, 1998.

[8] R. Negulescu, A.M.G. Peeters. "*Verification of speed-dependencies in single-rail handshake circuits*". In Proceedings of the Fourth International Symposium for Advanced Research in Asynchronous Circuits and Systems, 1998.

[9] R. Negulescu. "*Event-driven verification of switch-level correctness concerns*". In Proceedings of the International Conference on Application of Concurrency to System Design, 1998.

[10] R. Negulescu. *"A technique for finding and verifying speed-dependencies in gate circuits"*. In ACM International Workshop on Timing Issues in the Specification and-Synthesis of Digital Systems, 1997.

[11] X. Kong, R. Negulescu, L. Ying. *"Refinement-based formal verification of asynchronous wrappers for independently clocked domains in systems on chip"*. In Proceedings of the 11th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME), 2001.

[12] X. Kong, R. Negulescu. *"Formal verification of peephole optimizations in asynchronous circuits"*. In Proceedings of 21st IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), 2001.

[13] R. Negulescu. *"Generic transforms on incomplete specifications of asynchronous interfaces"*. Technical report, McGill University, 2003.

[14] R. Negulescu and R. T. Berks. *"Supervisory control for hierarchical verification of asynchronous circuits"*. Technical report, McGill University, 2003.

[15] P. Shirvani, S. Mitra, J. Ebergen, M. Roncken. DUDES: *"A fault abstraction and collapsing framework for asynchronous circuits"*. In Proceedings of the Sixth Symposium on Advanced Research in Asynchronous Circuits and Systems, 2000.

[16] R. Negulescu. *"A technique for finding and verifying speed-dependencies in gate circuits"*. Technical report CS-97-28, Dept. of Computer Science, University of Waterloo, Canada, 1997.

[17] Jo Ebergen, Charles Molnar, Radu Negulescu, Huub Schols, Bob Sproull (editor), Jan Tijmen Udding, Tom Verhoeff. *"And/if: a file format for exchanging finite automata descriptions"*. http://edis.win.tue.nl/and-if

Tool/Methodology: Handshake Technology Design Flow

Developer: Ad Peeters et al.

Organisation: Handshake Solutions, Philips Technology Incubator, Eindhoven, The Netherlands

Summary

The Handshake Technology design flow is based on the proprietary programming language Haste (formerly known as 'Tangram'), and uses handshake circuits as central representation format and intermediate architecture.

The programming language Haste was inspired by CSP and Occam. Haste supports variables (both latches and flip-flops, with optional reset values), channels (broadcast and narrowcast, arbitrated and non-arbitrated), register files, embedded memories, etcetera. Program constructs include assignments and communication, and furthermore all common imperative language constructs such as various conditional and iteration operators.

In addition, Haste offers language constructs for parallelism, channel communication, communication through shared variables, and sharing of hardware blocks. Furthermore, instantiation of RAMs, ROMs, and register files is supported in the language. Several constructs are implemented to support the design of external interfaces, such as waiting for edges or conditions, the direct manipulation of external signals, and the reliable sampling of external signals. A powerful parameter mechanism for both functions and procedures is also part of the language.

Haste allows for modular compilation, and supports the inclusion of datapath logic blocks from other tools, e.g., complex multipliers obtained from module generators from other EDA vendors.

The handshake circuit format supports both handshake and non-handshake communication. Components are available that implement language constructs such as sequencing, parallelism, conditional commands, and loops. Variables (both latches and flip-flop variants), datapath operators (both handshake and non-handshake) and many other components are supported.

The third level of abstraction is the internal abstract netlist format, which covers all common logic functions, a variety of latches and flip-flops, many generalized C-elements, a number of mutual-exclusion operators, and delay elements.

Various backends (from handshake circuits to gate-level netlist) have been implemented, though not all of them are operational today. The four-phase single-rail backend is being used in combination with a range of standard-cell libraries in various CMOS and other technologies. Also, a synchronous implementation of handshake circuits is supported to facilitate mapping onto FPGAs for prototyping.

For the standard-cell libraries, dedicated asynchronous cells (such as C-elements, mutual-exclusion elements and delay elements) are typically not available as dedicted elements, but

rather are mapped onto combinations (networks) of standard gates, such as nands, nors and complex and-or-inverts.

Application Domain

The Handshake Technology design flow is not limited to a specific domain.

Current applications include several products in the wireless communication area, smartcards,and in-vehicle networks for automotive.

Typically, applications are those where the benifits apply most, and where performance is less of an issue than power consumption, electromagnetic emission, and ease of integration. The complexity of the applications is not limited by the design flow.

Use of Existing HDLS

Many commercial tools are applied in the Handshake Technology design flow, especially in the backend of the flow. Actually, the drive has been to minimize the development of new dedicated tools, and to re-use common synchronous tools where-ever possible. Depending on the customer's wishes, the Handshake Technology flow can be targeted to any specific EDA flow, whether standard (such as Cadence or Synopsys) or even in-house.

In particular, the Handshake Technology design flow employs standard tools for the following tasks:

1. Prototyping on synchronous FPGAs
2. Technology mapping onto standard-cell libraries
3. Static timing analysis of logic blocks to support the delay matching in the control
4. Test-pattern generation for the scan-testable netlists
5. Various backend tasks such as drive-strength fixing and balancing of local clocks and scan-test clocks

Category

Synthesis, silicon compiler

Test Strategy

The test solution for Handshake Technology circuits is based on synchronous scan-test. During test, the circuit is operated as a synchronous circuit, and support scan operation and evaluation. The Handshake Technology test tools transforms the asynchronous sequential elements into elements that can be controlled using clock signals, and generates remodel files that can be used to generate test-patterns for the scan-testable circuit thus obtained using standard synchronous test-pattern generation tools such as Mentor's Fastscan or Synopsys' Tetramax. The test coverage against stuck-at faults is complete (that is, 100% unless redundant datapath logic has been used) and results in a test quality that is equivalent to that obtained for synchronous circuits.

Naturally, full scan-testability comes at some area cost, and an alternative (with higher time to market) is to use functional testing. In that case, one could program scan-like functions for the datapath directly in Haste, to support the generation of high-quality functional patterns. Their coverage can be evaluated using commercial fault-coverage tools.

Current status of Tool

The Handshake Technology design flow is operational at several sites, and has

been used to design dozens of ICs that are on the market.

Current Activities

There is a continuous drive to improve the design flow based on customer requirements and on lessons learned in the field. The main focus is on extending the user base of the tools.

Maintainer

Handshake Solutions, especially its tool and technology development teams.

Availability

The design flow is available from Handshake Solutions since January 1st 2004. For information contact: sales@handshakesolutions.com. (Until 2003 it was Philips proprietary.).

Future Plans

It is the goal of Handshake Solutions to make a commercial success of Handshake Technology, and selling it to the semiconductor industry.

Demonstrators

During the development of Handshake technology, as part of the project at Philips Research, several demonstrator ICs have been designed and evaluated since 1986. The initial demonstrators were toy applications, and were used to streamline the design flow and address the accuracy of simulation models. Since 1992, all demonstrators have been based on specifactions from industry, that is, were taken from or geared towards real commercial products. For instance, several ICs for the DCC (Digital Compact Cassette) system and for a video-on-demand application have been developed and were successfully tested.

Since 1995, the tool set is operational at Philips Semiconductors, who have since then made all the silicon, mosty directly as products. Examples include the 80c51 microcontroller, a family of pager ICs, wireless telphony controller, game consoles, smart cards, and in-vehicle networking products. (See also the 'Philips Semiconductor' section in the Industry Report.)

References

For more information refer to *www.handshakesolutions.com*

Tool/Methodology: LARD

Developer: P. Endecott

Organisation: University of Manchester

Summary

LARD (Language for Asynchronous Research and Development) is a hardware description language developed for describing asynchronous systems - though little is specific to that purpose, so you could use it to describe synchronous systems if you wanted, or even as a general purpose programming language.

Strengths & Weaknesses

Comparison of LARD with VHDL:

The main problems with VHDL as a language for modelling asynchronous circuits are:

- VHDL lacks channel communication. As a result it is necessary to explicitly model the request and acknowledge signals used in inter-block communication.
- While VHDL allows parallel composition of processes, those processes must be internally sequential. Asynchronous systems designers often want to have reconverging parallelism at the statement level, and getting the effect of this in VHDL is very awkward.
- VHDL is a very complex language, so writing tools that do something useful to it in finite time would be challenging.

Comparison of LARD with Tangram:

Tangram solves all of the problems of VHDL described above. Unfortunately it does suffer from some problems of its own: -Tangram lacks some of the high-level programming features that users normally take for granted; in particular it doesn’t implement records. -Tangram is proprietary to Philips.

The other problems result from the fact that Tangram is exclusively a synthesis language: -We might want to model things at a more abstract level than is possible in Tangram. We want to be able to run simulations with very abstract models that are not synthesisable. -On the other hand we might want to model things at a lower level than is possible in Tangram, for example with explicit modelling of signals rather than channels. -Tangram only models time by back-annotating delays from the synthesised model. We want to be able to give explicit timing information in our models so that we can get performance estimates out of abstract high-level models.

Application Domain

LARD is a modelling tool for architectural analysis and comparisons. It is up to the user to interpret the results to consider effects on performance/power etc.

Use of Existing HDLs

none

Extent of Automation

Once a model description has been written, the tool flow is automated.

Categories

Simulators/Modelling tools

Design Flow and Commercial EDA Tool Requirement

LARD fits into the design flow in 3 places:

1. Abstract models can be simulated to determine the basic architecture.
2. The system is then either constructed by hand using Commercial CAD tools or using Balsa.
3. If the systems is synthesised using Balsa, then LARD is used to perform behavioural simulations of the Balsa code to verify functionality before the final synthesis stage. Once a circuit netlist has been obtained, LARD can be used to interface to the Timemill simulator to apply test stimuli to the circuit.

Test Strategy

LARD is a flexible modelling language and thus can be used to debug designs.

Current Status of Tool

Current activities

The tool is stable and has been used for a number of purposes.

Maintainer

No longer maintained

Tool Availability

freely available from *http://www.cs.man.ac.uk/amulet/projects/lard*

Future Plans

None

Significant Demonstrators

AMULET3i was modelled using LARD, including the core, DMA controller, MARBLE bus, RAM and ROM. LARD was used for test vector application to validate the MARBLE design. An asynchronous cache subsystem for AMULET3 has deen developed using LARD.

References

From Behavioural Models to Silicon: Tools for the AMULET3 Project [P.B.Endecott & S.B.Furber, unpublished] describing how LARD has been used for AMULET3, focusing on the test generation problems. *ftp://ftp.cs.man.ac.uk/pub/amulet/lard/async98.ps.gz*

Modelling and Simulation of Asynchronous Systems using the LARD Hardware Description Language. [Phil Endecott, ESM98] *ftp://ftp.cs.man.ac.uk/pub/amulet/papers/lard_esm98.ps.gz*

Behavioural Modelling of Asynchronous Systems for Power and Performance Analysis. [P.B.Endecott & S.B.Furber, PATMOS'98]
ftp://ftp.cs.man.ac.uk/pub/amulet/papers/lard_patmos98.ps.gz

AMULET3i Cache Architecture
http://www.cs.man.ac.uk/amulet/publications/papers/async01_ying.html

Tool/Methodology: MINIMALIST

Developers: Steven Nowick, Robert Fuhrer, Alexander Shapiro, Michael Theobald Tiberiu Chelcea, Luis Plana

Organisation: Columbia University, New York, NY, USA

Summary

MINIMALIST is a CAD package for the synthesis, optimization and verification of asynchronous ‘burst-mode’ controllers. Burst-mode is a commonly-used Mealy-type asynchronous specification style.

The focus of the tool is on technology-independent synthesis. It includes a number of highly-optimised algorithms for state minimization, optimal state assignment, two-level hazard-free logic minimization, synthesis for generalized C-element implementations, and verification. It also provides a graphic display of specifications and implementations, an interactive shell, design scripts, help menus, and a tutorial. The synthesized implementations are hazard-free gate-level circuits, either two-level (AND-OR) or using generalized-C elements. These circuits can then be technology-mapped using additional tools.

The key goal of the MINIMALIST package is to facilitate *design-space exploration*: for a given specification, the designer can synthesize a variety of different implementations, exploring trade-offs of performance and area. Many synthesis options are provided, to search for high-quality designs. These include several *machine styles* (using outputs as fedback state variables, or not, to reduce number of state bits); *logic implementation styles* (allowing various levels of sharing of logic between different outputs); *cost functions* (such as, optimising for number of products, literals, critical path, etc.); and *algorithmic modes* (exact vs. heuristic). As a result, dozens of different implementations can be synthesized for each specification, guided by user settings.

The package also includes a number of first-of-a-kind optimization algorithms, including: CHASM, for optimal state encoding (for example, it can find a critical race-free state encoding which produces minimum-cost hazard-free logic for outputs); and several different exact (hfmin, IMPYMIN) and heuristic (espresso-HF) two-level logic minimizers.

MINIMALIST can handle fairly large controllers efficiently, and has a number of heuristic switches to speed up synthesis. It provides several back-end formats, including PLA format and Verilog HDL. It also includes a verifier, which compares the specification to the gate-level implementation, checking for both functionality and hazard violations.

Strengths and Weaknesses

Strengths: Designs mappable to a variety of commercial standard libraries; Mealy-style specifications which are comfortable for synchronous designers; handle reasonably large controllers with good run-time; excellent support for design-space exploration; includes several exact optimization algorithms (including state assignment and logic minimization).

Weaknesses: currently does not include technology mapping, an HDL front-end, or support for ‘extended burst-mode’. (All of these features are planned for future additions to the tool; several independent algorithms and design tools exist for burst-mode technology mapping, but

are not yet incorporated into MINIMALIST.) Moderate one-sided timing assumptions, which require verification and possible delay-padding (off of critical paths, usually).

Application Domain

Medium to high-performance control circuits. Use moderate timing assumptions (one-sided); neither overly aggressive nor delay-insensitive.

Use of Existing HDLs

Implementations can be written out in several formats, including Verilog HDL, which can then be handed to external tools for backend (technology-dependent) synthesis.

Extent of Automation

Push-button synthesis, from controller specification to gate-level implementation, using pre-packaged scripts.

Category

Synthesis – Burst-mode.

Design Flow and Commercial EDA Tool Requirement

MINIMALIST is used for synthesis and optimization of individual asynchronous controllers. The designer invokes the MINIMALIST shell, which is an interactive environment for design. MINIMALIST commands can be invoked from the shell, as well as useful Unix commands (e.g., ls, cd, etc.).

The input to the tool is a burst-mode specification, in text “bms” file format. The designer has many options for performing synthesis runs (see below). However, in all cases the basic steps are as follows:

- state minimization;
- state assignment (i.e., state encoding);
- two-level logic minimization (or mapping to generalized-C elements);
- verification.

A number of design options are provided, to search for high-quality designs: machine style (using outputs as fedback state variables, or not); logic implementation style (various levels of sharing of logic between different outputs); cost functions (optimizing for number of products, literals, critical path, etc.); and algorithmic modes (exact, heuristic).

The designer can interact with MINIMALIST in two modes: (a) using prepared ‘design scripts’, or (b) typing individual ‘commands’ for more detailed control over the synthesis steps. For (a), there are a variety of scripts: several do a complete prepackaged synthesis run, oriented either to a basic implementation, or to optimization for speed or area. Others do a set of 4 runs, so the user can pick the best of the set. For (b), the experienced user can control settings on each individual step of synthesis, by typing commands directly to the shell.

Test Strategy

Synthesis-for-testability methods for MINIMALIST have been developed and published, but are not currently incorporated into the tool. MINIMALIST does provide several output formats for gate-level circuit implementations, including Verilog HDL, which can be simulated.

Current Status of Tool

Current Activities

Release 1.2 will be available by end of July 2001, including many new features: mapping to generalized-C elements, tutorial support, graphical display of specifications and circuit implementations, design scripts, verification, a Verilog HDL back-end, and several advanced features.

Maintainer

The contact person is: Steven Nowick (nowick@cs.columbia.edu). The maintainers are: Steven Nowick (nowick@cs.columbia.edu) and Robert Fuhrer (rfuhrer@watson.ibm.com).

Availability

Freely available for non-commercial use at: *http://www.cs.columbia.edu/async*. Available through license for commercial use; send email to contact person above (Steven Nowick) for inquiries.

Future Plans

Future releases are expected with extensions to handle “extended burst-mode (XBM)” specifications. Potential incorporation with Balsa and other tools as a technology-independent back-end. Addition of simulation and verification tools.

Demonstrators

Portions of the MINIMALIST tool (hazard-free logic minimization tools) have been included in the 3D package and applied to several experimental designs: (a) Hewlett-Packard “Stetson” project, for design of infrared communications chip; (Alan Marshall and Bill Coates and Polly Siegel, “Designing an Asynchronous Communications Chip”, IEEE Design and Test of Computers, 11:2, pages 8-21, 1994); (b) Intel “RAPPID” instruction-length decoder (S. Rotem et al., “RAPPID: an asynchronous instruction-length decoder”, Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Computer Society Press, pages 60-70, April 1999).

Tool/Methodology: Oolong

Developer: Will Toms

Organisation: University of Manchester, UK

Summary

Oolong is a technology-independent synthesis tool capable of synthesising arbitrary QDI/SI combinational logic functions. Oolong employs traditional Multi-level logic synthesis techniques constrained to uphold the SI requirements of the resulting circuits.

Return-to-Zero DI-encoded combinational functions form an interesting subset of the set of SI circuits. The excitation-regions of the function outputs overlap and contain many common signals. For this reason traditional SI decomposition/synthesis techniques[] that generate candidate divisors from individual functions and then evaluate their cost in the whole network, can suffer as valid decompositions for individual signals may greatly increase the cost of decomposing other signals. Oolong uses the matrices of multi-level logic synthesis to determine sets of signals that may be inserted into an implementation without violating SI/QDI constraints and how they affect all the output signals of the network. The matrix models may be used to successfully determine candidate divisors - but the set of divisors must be severly constrained to maintain the ability of the matrix model to determine all relationships between signals in a network. This can be reduced by incorrectly selecting divisors that bind sub-terms to inserted signals and mask common sub-terms.

Oolong has been encorporated into the Balsa synthesis system to allow the synthesis of arbitrary encoded QDI circuits. The circuits produce contain only *Isochronic fork* assumptions and hence are QDI although as synthesised circuits may consist of hundreds of gates a Speed-Independent moniker may be more suitable.

Strengths and Weaknesses

Strengths.

- Oolong can synthesise a large-range of DI-encoded QDI circuits, including many that were previously unsynthesisable in an implementable form (i.e. without unlimited fan-in gates).

- All circuits are technology independent (assuming a basis of 2-input C-elements and Or-gates) and hence can be mapped to most libraries.

- Circuits are synthesised using existing multi-level logic techniques that allow multi-cube divisors.

Weaknesses.

- Oolong implements strongly-indicating functions and so implementations may be large particualrly for functions involving m-of-n codes where m is greater than 4. For such circuits the synthesis process may never terminate, and so no realistic strongly indicating function is possible.

- As a proof of concept and to fit in with the existing Balsa framework, Oolong has been implemented in Scheme - an interpreted language, the code is fairly inefficient and hence for large circuits (> 500 minterms) the synthesis times may be several hours.

Application Domain

Synthesis of DI-encoded QDI/SI Combinational Logic Circuits.

Use of existing HDLs

Oolong has been incorporated into the Balsa system, although is a stand-alone low-level circuit synthesis tool.

Extent of automation

Descriptions are provided to Oolong using a two-level blif format such as those used in Espresso. The output is a multi-level blif file. Oolong can also output an internal format suitable for the balsa system. It is intended to incorporate the technology description and library declaration system of Balsa to allow oolong to produce technology specific netlists, that may be simulated.

Category

QDI/SI combinational logic synthesis.

Design Flow and Commercial EDA Tool Requirement

The Oolong stand-alone tool currently just inputs and outputs blif files. As part of the Balsa synthesis framework Oolong is used to synthesis m-of-n QDI logic circuits into CAD-specific netlists, see the Balsa submission in this report.

Test strategy

None, although there is a simple SI verifier which may be made available.

Current status

Maintainer

Will Toms (tomsw@cs.man.ac.uk

Availability

Not currently available but will provided on request to maintainer.

Future plans

To implement functions efficiently in e.g. C, to reduce synthesis time. To implement weak-conditioned logic to reduce circuit size, latency.

Demonstrators

Oolong has been used in the Balsa system to synthesise a m-of-n encoded 32-bit MIPS core. Further examples using significant balsa circuits are planned.

Tool/Methodology: OptiMist

Developer: Danil Sokolov

Organisation: School of Electrical, Electronic and Computer Engineering, University of Newcastle, Newcastle upon Tyne, UK

Summary

OptiMist (Optimise and Map) is a package of tools that optimise Signal Transition Graph (STG) specifications and map them into asynchronous circuits. The package works in 4 stages:

Output exposure. At this stage the initially closed (with both input and output transitions) system specification is transferred into an open system specification using the concept of environment tracking and output exposure [1]. The resultant specification consists of two blocks: a tracker and a bouncer. The tracker follows the behaviour of the environment and generates context signals for the bouncer. The bouncer is a set of elementary cycles, each of which represents an output signal. An elementary cycle consists of two places, corresponding to low and high levels of the signal, and positive and negative transitions of the signal. The elementary cycle switches between low and high states using a signal from the tracker.

Detection of redundant places. When outputs are exposed some places in the tracker become redundant, because an output can be switched by directly preceding input signal without using intermediate state of the tracker. Many tracker places can thus be removed, provided that the system behaviour is preserved w.r.t. input-output interface (weak bisimulation). Their elimination is however restricted by potential coding conflicts (which may cause tracker errors) and implementation constraints (at least three DCs in every loop). Redundant places are detected using a set of optimisation heuristics [4]. Initially all places are not tagged. Then every place is given a tag: redundant (if the place can be removed) or mandatory (if the place is

necessary). The heuristics define the order in which the places should be checked.

Heuristic A (latency reduction): Places after input but before output transitions are tagged as redundant.

Heuristic B (size reduction): The chains of places between redundant places detected in the previous heuristic are considered. The places of each chain are tagged starting from the beginning of the chain and going in the direction of consuming arcs. The last place in each chain is not tagged.

Heuristic C (size reduction): All non-tagged places (the last places in the chains) are tagged individually.

Elimination of redundant places. At this stage redundant places are removed from the tracker one by one. After a removal of each place the specification is modified preserving the behaviour of the system. If a place has only one preceding transition and one succeeding transition then the modification is trivial (the place is deleted and the conjugate transitions are replaced by one). In case of more then one preceding or succeeding transitions, more sophisticated transformation is required for place removal.

Sometimes the number of context signals (signals from tracker) for an elementary cycle is increasing after removal of a redundant place. This can be explained by the splitting of a mandatory place before the deep hierarchy of forks and the recalculation of context signals from redundant places to the split places. The solution to this problem is to evaluate the complexity of elements by calculating the maximum number of incoming arcs into a transition before and after removal of every place. If this number is increasing (or at least is increasing beyond the maximum implementable value), then the place should be kept in the specification. This is the element complexity optimisation technique which reduces the size of the circuit by preventing complication of logic rather than reducing the number of state holding elements.

Mapping of into circuit. Finally, the places of the tracker are mapped into David Cells (DCs) and the elementary cycles are mapped into set-reset flip-flops (FFs). The netlist of DCs and FFs is generated in Verilog format.

Strengths and Weaknesses

Strengths:

- Generated circuits have low output latency.
- All optimisation is performed at the specification level as opposed to optimisation of logic circuits after the stage of synthesis.
- Tool can process large specifications (which are not computable by logic synthesis tools in acceptable time), because optimisation is performed locally and the computation time grows almost linear with the size of specification.
- The tools are extremely fast, which allows the designer to try different combinations of heuristics and different start points of optimisation.
- The element complexity optimisation facilitates technology mapping by restricting the growth of logic element fanin.

Weaknesses:

- For small and medium examples the circuits produced by the proposed technique are usually larger than the solutions of logic synthesis tools.

Application Domain

High-speed asynchronous controllers.

Use of Existing HDLs

The tool takes STG in ASTG format and generates a netlist of DCs and FFs in Verilog format.

Extent of Automation

The tools are fully automated. At the same time a designer can significantly influence on the result at the stage of redundant place detection by choosing one or more of the proposed heuristics.

Category

Synthesis - STG/Petri net.

Design Flow and Commercial EDA Tools

OptiMist package can be employed in combination with Cadence to allow simulation and technology mapping of circuits. A basic library of DCs and FFs has been created for Cadence. It can be expanded, if necessary, using a tool from the package which generates a Verilog implementing DCs and FFs at transistor level.

Test Strategy

N/A

Current Status of the Tool

Current Activities

The tool is currently in testing stage.

Maintainer

Danil Sokolov (*danil.sokolov@ncl.ac.uk*)

Availability

http://async.org.uk/

Future Plans

- To improve the algorithm of redundant place detection to avoid growth of DCs/FFs fanins.
- To try several start points in redundant place detection procedure and automatically select the best result.

Demonstrators

N/A

References

[1] A.Bystrov and A.Yakovlev, "*Asynchronous circuit synthesis by direct mapping: Interfacing to environment*", In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems, pages 127-136, Manchester, UK, April 2002.

[2] D.Sokolov, A.Bystrov and A.Yakovlev, "*Automated design of low-latency asynchronous circuits by direct mapping"*, Postgraduate Research Conference in Electronics, Photonics, Communications and Software, Notingham, UK, April 2002.

[3] D.Sokolov, A.Bystrov and A.Yakovlev, "*Tools for STG optimisation in the direct mapping of asynchronous circuits"*, 12th UK Asynchronous Forum, London, UK, June 2002.

[4] D.Sokolov, A.Bystrov and A.Yakovlev, "*STG optimisation in the direct mapping of asynchronous circuits"*, In Proc. of Design Automation and Test in Europe, Mart 2003.

Tool/Methodology: Petrify

Developer: Jordi Cortadella

Organisation: Universitat Politenica de Catalunya, Spain

Contributors: Michael Kishinevsky (Intel Corporation, USA) Alex Kondratyev (Cadence Design Systems, USA) Luciano Lavagno (Politecnico di Torino, Italy) Enric Pastor (Universitat Politecnica de Catalunya, Spain) Alexander Taubin (Boston University, USA) Alexandre Yakovlev (University of Newcastle upon Tyne, UK)

Summary

Petrify is a tool the for synthesis of Petri nets and asynchronous controllers. The tool can read a Signal Transition Graph (STG) and generate a circuit netlist. An STG is a formal specification of timing diagrams in which the events correspond to rising and falling transitions of control signals. Besides expressing causality, STGs can also specify concurrency and choice. During synthesis, petrify solves several logic synthesis problems required to produce a netlist: state encoding, logic decomposition and technology mapping.

Petrify can either synthesize speed-independent circuits or timed circuits. In the latter case, the designer can provide relative timing assumptions under which the correctness of the system can be guaranteed. Petrify backannotates the required timing constraints that must be verified after synthesis. Petrify can provide different types of netlists: Boolean equations, generalized C-elements or gates from a library. Additionally, set/reset points in the circuit are detected and information about their initialization is given.

As a final step, petrify can also synthesize STGs with the purpose of backannotating the transformations performed during synthesis. In this way, all the internal signals inserted for encoding and logic decomposition, can be observed in the same format as the designer has specified the system.

Petrify also includes a tools to visualize STGs and State Graphs (draw_astg and write_sg).

Strengths and Weaknesses

The tool provides fully automation during synthesis. The used of symbolic techniques to represent the state space allows to synthesize large controllers (more than 20 signals in case of regular well-structured behaviours). High-quality circuits are obtained.

Not appropriate for data-path synthesis. Obtaining a circuit netlist cannot always be guaranteed.

Application Domain

High-speed asynchronous controllers.

Use of existing HDLs

Netlist can be generated in Verilog, BLIF or EQN. Gate libraries are read in genlib format (from SIS).

Extent of automation

Complete automation from STG to circuit netlist.

Category

Synthesis - STG/Petri net

Design Flow and Commercial EDA Tool Requirement

Petrify can be used as a standalone tool for synthesis. The design flow starts from a specification of the system's behaviour described by an STG. A textual format is used for such specification. Additionally, a gate library must be provided in case a mapped netlist is desired after synthesis. A push-button approach is provided to generate the circuit.

Test Strategy

No test vectors are generated.

Current Status of the Tool

Current Activities

Maintenance for users

Maintainer

Jordi Cortadella (jordic@lsi.upc.es)

Availability

http://www.lsi.upc.es/~jordic/petrify

Future plans

The team is working in a second generation of the tool aiming at the synthesis of large and well-structured controllers. There are plans to integrate the new tool in a high-level synthesis framework.

Demonstrators

The tool has been used by different groups for the synthesis of asynchronous controller. Here are some examples:

- Several circuits from RAPPID (Intel Corp.) have been synthesized. Circuits with similar quality to those designed manually have been automatically generated.
- Several controllers for the AMULET microprocessor have been synthesized.
- Theseus Logic is using the tool for the synthesis of controllers.

Tool/Methodology: Phased Logic

Developers: Robert B. Reese, Mitch A. Thornton & Cherrice Traver

Organisations: Mississippi State University, Southern Methodist University and Union College

Summary

Phased Logic (PL) is a self-timed design methodology that provides an automated translation of a clocked system in the form of D-flip-flops and combinational gates to a self-timed netlist of PL gates. The only global net in the self-timed netlist is a reset signal. The PL netlist is a micropipelined system with two-phase control. Two distinct implementation technologies are supported, *fine-grain* and *coarse-grain*. The fine-grain approach uses a one-to-one mapping of gates in the clocked system to PL gates that use a 4-input Lookup-Table (LUT4) as the logic element with delay-insensitive dual-rail routing between gates. This technology could form the basis for the implementation of a self-timed FGPA. Because all routing between gates is delay-insensitive, there are no timing mechanisms external to a PL gate that can cause a failure due to timing. The coarse-grain approach maps groups of gates in the clocked netlist to the combinational compute function of a PL block, with bundled data signaling used between blocks. The combinational compute function of a coarse-grain PL block can be implemented using a traditional standard cell library. The coarse-grain technology is an ASIC approach to the implementation of PL systems. All timing concerns in a coarse-grain implementation are block-to-block; there are no global mechanisms that can cause failure due to timing. If desired, delay insensitive signaling can be used between coarse grain blocks to remove timing uncertainty due to wire delays. This will add extra latency in the control path but this latency can be hidden if the coarse grain block delay is long enough. Both fine-grain and coarse-grain approaches support a speedup mechanism known as *early evaluation* (a generalized form of bypass) that can allow the PL system to outperform the clocked system in some cases. All micropipeline approaches suffer a performance penalty compared to clocked systems because the output latch latency of a micropipeline block is in the critical path. Early evaluation allows PL systems to recover some of this performance penalty.

Strengths and Weaknesses

Strengths: Automated flow from clocked gate-level netlist; we use Synopsys Design Compiler and either VHDL or Verilog to generate the starting designs. The resulting self-timed design can outperform the clocked design in some cases when early-evaluation is used appropriately. Early evaluation identification is automated in the fine-grain flow.

Weaknesses: No silicon demonstrated yet; coarse-grain flow has been demonstrated down to gate-level Verilog netlists of two commercial standard cell libraries (Artisan, UMC) with pre-layout SDF timing. The fine-grain flow is only implemented using behavioral models for the fine-grain gates. There is currently no support for multiple clock domains in the starting clocked netlist. The coarse-grain flow requires top level partitioning of the design into blocks by the designer, and the designer must manually identify early evaluation opportunities.

Application Domain

The coarse-grain flow assumes a static CMOS gate library, so medium performance designs are targeted. There is no inherent limit on design size.

Use of Existing HDLs

The starting point is a gate-level netlist, so any existing HDL /synthesis tool can be used to produce this netlist. The coarse-grain flow is tied to Synopsys Design Compiler for static timing analysis. We have used VHDL/Verilog and Design Compiler to produce starting netlists.

Category

Fine-grain flow: two-phase micropipeline, Level-encoded Dual Signaling (LEDR) used between gates for delay-insensitivity; no distinction between datapath and control in clocked netlist.

Coarse-grain flow: two-phase micropipeline, bundled-data signaling used between blocks; no distinction between datapath and control in clocked netlist.

Design Flow and Commercial EDA Tool Requirement

See figure for coarse-grain flow; this flow is tied to Synopsys Design Compiler which is used for static path analysis. The fine-grain flow has only been demonstrated with Synopsys DC, but could use any front-end synthesis tool to produce the starting netlist that contains DFFs and 4-input lookup tables (LUT4s).

Test Strategy

Clocked scan-paths in the original clocked netlists are preserved in both fine-grain and coarse-grain PL netlists.

Current Status of Tool

Tools are available, see *http://www.ece.msstate.edu/~reese*, follow the Phased Logic tool link. Binaries supplied for both SUNOS/Sparc and Linux/X86 platforms.

Maintainer

Robert B. Reese; reese@ece.msstate.edu

Future Plans

We are currently working on a silicon demonstration for the coarse-grain flow. A back-end implementation for the fine-grain flow is under development.

Demonstrations

A tutorial with sample designs for both fine-grain and coarse-grain flows was presented at ASYNC 2004. The examples include a 5-stage pipelined CPU for the coarse-grain flow. The tutorial and examples are available with the tool distribution.

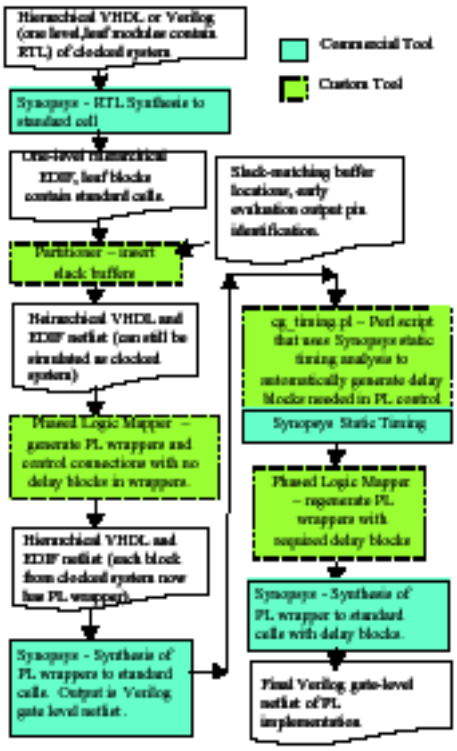

Figure 1: Phased-Logic Tool Flow

References

[1] Linder, D.H. and Harden J.C. "*Phased Logic: Supporting the Synchronous Design Paradigm with Delay-insensitive Circuitry*", IEEE Transactions on Computers, Vol. 45, No 9, (1996)

[2] Reese. R.B., Thornton, M.A., and Traver, C. "*A Coarse-grained Phased Logic CPU*", Ninth International Symposium on Advanced Research in Asynchronous Circuits and Systems (2003).

[3] M.A. Thornton, K. Fazel, R.B. Reese, and C. Traver, "*Generalized Early Evaluation in Self-Timed Circuits*", Proceedings of DATE 2002, Paris France, March 4-8, 2002.

[4] Reese. R.B., Thornton, M.A., and Traver, C. A Fine-grain Phased Logic CPU. In Proceedings of the IEEE Computer Society Annual Symposium on VLSI (2003).

[5] Reese, R.B., Thornton, M.A., and Traver, C. "*Arithmetic Logic Circuits using Self-timed Bit-Level Dataflow and Early Evaluation*", In Proceedings of the International Conference on Computer Design (2001).

Tool/Methodology: PipeFitter

Developers: I. Blunno, L. Lavagno & V.P. Shah

Organisation: Politecnico di Torino

Summary

Pipefitter is a tool for the automated synthesis of micro-pipelined asynchronous circuits, with a 4-phase control unit supporting concurrency, sequencing and choice, and a synchronous data path with matched delays.

Strengths and Weaknesses

The main advantages are that a complete asynchronous design flow is supported and that the specification language is standard behavioural Verilog. The main disadvantage is that at this time only a few Verilog statements are supported and the synthesis process is only partially automated.

Application Domain

Low electro-magnetic emissions ASIC design.

Use of Existing HDLs

The specification language is Verilog HDL, 100% consistent with the simulation semantics

Extent of Automation

This, the first version of the new pipefitter is intended for simulation purposes only and no guarantee is given on the hazard freedom of any component if synthesis is performed even though the expert designer can try to properly constrain this process. So far only high-level synthesis is implemented by pipefitter (i.e., asynchronous verilog to synthesisable verilog for both data path and control unit). Future versions of pipefitter will support the automated synthesis through the generation of proper synthesis scripts. Designers can also manually modify intermediate representations (e.g., the synchronous data path, written in synthesizable Verilog).

Category

Synthesis

Design Flow and Commercial EDA Tool Requirement

The input is a specification written in an “asynchronous synthesizable subset” of Verilog, including RTL-like constructs, and based on the Verilog simulation semantics. From this specification pipefitter automatically derives several Verilog netlists: one for the control unit and the other for each register and operative unit in the data path. The control unit is directly implemented, using an approach close to Hollaar’s (also called David cell-based).

Each data path module is synthesised into standard cells by an appropriate RTL/logic synthesis tool, such as the Synopsys Design Compiler or Cadence RC or Mentor Leonardo. Timing analysis must then be performed on the data path, in order to determine the worst case delays.

Its result can be used to modify the automatically generated matched delay block for each register and data path block, in order to generate the proper acknowledge signals for the control unit. This step will be automated by the generation of scripts in the future versions of pipefitter.

Test Strategy

The tool produces a Verilog which describes the whole synthesised system (Control Unit + Data Path). Synchronous scan chains can be used to test the data path. The controller can also be tested by a scan-based approach, but no support is currently provided for this step.

Current Status of Tool

Current activities

Pipefitter now is in the process of being completely redesigned and improved with new features. The first version of the new pipefitter can still be used for simulation purposes and for logic synthesis as well, even though for the latter the designer will have to manually constrain the process.

Maintainer

Pipefitter (preferred): pipefitter@gandalf.polito.it

Ivan Blunno: blunno@polito.it

Luciano Lavagno: lavagno@polito.it

Vishal P. Shah: vishal@gandalf.polito.it

Tool Availability

The tool is free. Users are encouraged to download it and provide the maintainers with feedback. The tool, the manual and more information are available from:

http://polimage.polito.it/~pipefitter

Future Plans

Extension of the supported verilog and automated generation of the scripts to drive timing analysis and synthesis.

Significant Demonstrators

The core of an asynchronous DLX has been designed with an older version of pipefitter. The new version still doesn't support all the Verilog statements supported in the previous versions and only simple arithmetical units have been designed so far.

References

[1] M.Amde, I.Blunno, C.P.Sotiriou, "*Automating the Design of an Asynchronous DLX Microprocessor*", Design Automation Conference 2003.

[2] I.Blunno and L.Lavagno, "*Designing an asynchronous microcontroller using Pipefitter*", IEEE International Conference on Computer Design: VLSI in Computers & Processors, September 2002.

[3] I. Blunno & L. Lavagno, "*Automated Synthesis of Micro-Pipelines from Behavioural Verilog HDL*", IEEE 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems, Israel, April 2000.

Tool/Methodology: Punf

Victor Khomenko

School of Computing Science, University of Newcastle upon Tyne

Summary

Punf [2,3,5,7] is a Petri net unfolder, i.e. it takes a Petri net (e.g. an STG) and produces a finite complete prefix of its unfolding. Such a prefix is a concise representation of a Petri net's state space and can be used for efficient model checking and synthesis. For STGs such a representation is often superior to that based on explicit state graphs or BDDs due to the fact that STGs usually contain a lot of concurrency but rather few choices. As a result, the memory requirements of synthesis algorithms based on unfoldings are very moderate.

Strengths and Weaknesses

Strengths. fast and memory efficient., supports STGs, low-level and high-level Petri nets.

Weaknesses. currently only safe nets are supported.

Application domain

Synthesis of self-timed circuits and formal verification.

Use of existing HDLs

none (it uses the standard `.g' STG format) [1].

Extent of automation

fully automatic.

Category

Synthesis and verification => STG/Petri net

Design flow

Punf is intended as a powerful unfolding engine to be used by other applications. Currently, prefixes produced by Punf can be used by the Clp, VerySAT and ConfRes [4,6,8,9.10] tools (described in this report) for detection of coding (CSC and USC) conflicts in STGs and for deriving a complex-gate implementation of a circuit. These tools comprise a complete framework for complex-gate synthesis of speed-independent circuits from STGs.

Test strategy

prefixes produced by Punf can be used by the Clp, VerySAT and tools within Pep for verification of various safety properties such as deadlock-freeness, mutex, reachability, etc.

Current status

Punf is fully operational. It is integrated into Pep[1]. Punf can unfold STGs, low-level and high-level Petri nets. Support for verification of LTL-X properties is to appear soon.

Maintainer

Victor Khomenko (*Victor.Khomenko@ncl.ac.uk*).

Availability

available for research purposes from

http://www.cs.ncl.ac.uk/people/victor.khomenko/home.formal/tools/tools.html

Future plans

we plan to create a full design cycle for self-timed circuits based on Petri net unfoldings and not involving building the state space at any stage.

References

[1] E.Best and B.Grahlmann: *"PEP: Documentation and User Guide, Version 1.4. Manual"* (1995).

[2] K.Heljanko, V.Khomenko, and M.Koutny: *"Parallelisation of the Petri Net Unfolding Algorithm"*. TACAS'2002, LNCS 2280 (2002) 371-385.

[3] V.Khomenko: *"Punf: Documentation and User Guide, Version 6.01. Manual"* (2002).

[4] V.Khomenko: Clp: *"Documentation and User Guide. Version 3.01beta. Manual"* (2002).

[5] V.Khomenko and M.Koutny: *"Towards An Efficient Algorithm for Unfolding Petri Nets"*. CONCUR'2001, LNCS 2154 (2001) 366-380.

[6] V.Khomenko, M.Koutny, and A.Yakovlev: *"Detecting State Coding Conflicts in STGs Using Integer Programming"*. DATE'2002, IEEE Comp. Soc. Press (2002) 338-345.

[7] V.Khomenko and M.Koutny: *"Branching Processes of High-Level Petri Nets"*. TACAS'2003, LNCS (2003)

[8] V.Khomenko, M.Koutny and A.Yakovlev: *"Detecting State Coding Conflicts in STG Unfoldings Using SAT"* Fundamentae Informatica, Sepecial Issue on Best Papers from ICACSD'2003 (to appear).

[9] V.Khomenko, M.Koutny and A.Yakovlev: *"Logic Synthesis Avoiding State Space Explosion"* ICACSD '04. (2004 to appear).

[10] A.Madalinski, A.Bystrov, V.Khomenko, and A.Yakovlev: *"Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design"*. IEE Proceedings: Computers & Digital Techniques: Special Issue on Best Papers from DATE'2003.(2003)

Tool/Methodology: SIS

Developers: L. Lavagno, Choo Moon, Paul Stephan

Organisation: University of California at Berkeley

Summary

SIS in general is a state-of-the-art logic synthesis tools, which includes a number of packages that are tailored to synthesize asynchronous circuits. Several delay models and synthesis algorithms, including design-for-testability considerations, are supported. The specification formalism is that of Signal Transition Graphs, and the output is a gate-level netlist that can be translated into Verilog and VHDL.

Strength and Weaknesses

The main advantage is the modularity of the tool, that makes it very easy to build on existing packages to add new algorithms and flows. A full set of algebraic and Boolean optimisations are already provided, including technology mapping, as well as I/O packages to read and write popular standard formats (STG, BLIF, Verilog and VHDL gate-level netlists).

The main disadvantage is that development of the asynchronous synthesis algorithms stopped around 1994 (some of the authors moved to the Petrify team).

Application Domain

Asynchronous control units.

Use of Existing HDLs

A translator from the output netlist into VHDL and Verilog is supported. The input format is compatible with one of the outputs of Pipefitter.

Extent of Automation

Fully automated logic synthesis from a Signal Transition Graph specification (interpreted Petri Net). Includes state encoding, logic minimization, design-for-testability (based on partial scan for sequential elements) and technology mapping.

Category

Synthesis, STG/PetriNet

Design Flow and Commercial EDA Tool Requirement

See above. No links to commercial tools are provided, except for the back-end.

Test Strategy

A translator from the output netlist into VHDL and Verilog is supported. The input format is compatible with one of the outputs of Pipefitter.

Current Status of Tool

Current activities

The tool is no longer being actively developed. Some bug fixing may be possible.

Maintainer

The SIS team: *sis@ic.eecs.berkeley.edu*

Tool Availability

The tool and the user's manual are available from: *ftp://ic.eecs.berkeley.edu/pub/Sis*

Future Plans

see above

Significant Demonstrators

Not stated

References

L. Lavagno and A. Sangiovanni-Vincentelli. "Algorithms for synthesis and testing of asynchronous circuits." Kluwer Academic Publishers, 1993.

E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton and A.L. Sangiovanni-Vincentelli. "Sequential Circuit Design Using Synthesis and Optimization". Proceedings of the International Conference on Computer Design, October 1992

Tool/Methodology: TAST

Developer: CIS group

Organisation: TIMA Laboratory

Summary

TAST (Tool for Asynchronous circuits SynThesis) is a compiler generating different kinds of asynchronous circuits from a high level description. Circuits are modelled using a CSP-like language, called CHP (Communicating Hardware Processes), enriched with statements useful for simulation and synthesis purposes. The compiler is general enough to accept several input languages (front end) and target several logic styles (back end). Input programs are first analysed and checked for some properties by the front end, such as concurrent assignments of variables, channel declarations and communication actions coherency, types compatibility, etc. When the verification phase do not exhibit errors, the program is transformed into an internal form based on Petri Nets and Data Flow Graphs.

At this stage, the programmer/designer can choose to generate a functional VHDL description of the model. This VHDL description can be used to simulate the design. An option also enables the designer to generate an RTL VHDL description, which can be used to target ASICs or FPGAs technologies by means of standard CAD tools. The programmer/designer can also decide to target asynchronous circuits. The compiler then analyses the internal form to check whether the models can be mapped onto asynchronous digital circuits or not. To be accepted for synthesis, CHP programs have to respect some programming rules which ensure a correct mapping of the internal form into asynchronous circuits. If it is the case, micropipeline or QDI is chosen as a target and a VHDL gate netlist is generated. The produced gate netlist can of course be simulated using standard CAD tools.

Strengths and Weaknesses

- Data encoding is defined at the language level. General “1 out of N” codes are available.
- Handshake protocols can easily be modified.
- QDI and micropipeline (or mixed) circuits are supported as an option for output circuits.
- Co-simulation of functional description, gate level synchronous/asynchronous circuit netlists is supported using VHDL.
- Micropipeline control logic and QDI logic could be optimised further.
- Timing analysis/evaluation is not addressed yet
- Testability is not addressed yet.

Application Domain

- High complexity low to high speed asynchronous circuit design, both micropipeline and QDI styles.

- Mixed synchronous and asynchronous circuits.

Use of Existing HDLs

Any commercial VHDL simulation tool (Synopsys, Mentor...) can be used to simulate the designs (pre- and post-synthesis). Standard CAD tools can be used for Place and Route.

Extent of Automation

- Easy to set up, Unix/Linux platform supported.
- Produced VHDL models are compatible with commercial CAD tools.
- Command-line approach allows the use of scripts.

Category

Synthesis, silicon compiler

Design Flow and Commercial EDA Tool Requirement

VHDL translation is an improved version of the previously developed CHP2VHDL tool. In particular, it now supports the parallel operator without any restriction. The VHDL code generator brings all the features and facilities of VHDL to the designer: verification of CHP models, co-simulation of functional and gate-level descriptions, co-simulation of synchronous and asynchronous circuit models. A new feature enables the generation of a clocked RTL description for early prototyping.

Some further verifications are performed to ensure that input programs respect the synthesisable CHP. Two choices of circuit model are supported: QDI and micropipeline. In case that micropipeline model is selected, two VHDL models are created: a gate netlist describing the control part of the circuit and a data-flow description for the data-paths of the circuit. The latter is ready for synthesis using standard commercial synthesis tools.

Test Strategy

The CHP to VHDL translation tool enables the designer to develop a complete test environment in CHP. The test bench can also be developed in pure VHDL using a package provided with the tool suite. The package includes different kinds of functions such as send data to, receive data from channels, probe channels, together with a set of type-conversion functions.

The same test bench can be used for functional and post-synthesis simulations by means of VHDL configurations.

Current Status of Tool

TAST currently enables the design of significantly large circuits. However the optimization of the circuits at the gate level is still being improved in order to reduce the circuit complexity and area.

Maintainer

CIS team at TIMA. Contact: Marc Renaudin *marc.renaudin@imag.fr*

Tool Availability

Not available yet.

Future Plans

Support other input languages such as VHDL, Verilog or SystemC. Automatic translation of CHP into C programs to enable very fast functional simulations and generate reference simulation models of the circuits. Optimization of the produced asynchronous circuits at the gate level.

Significant Demonstrators

Aspro and Mica processors were designed using preliminary versions of TAST.

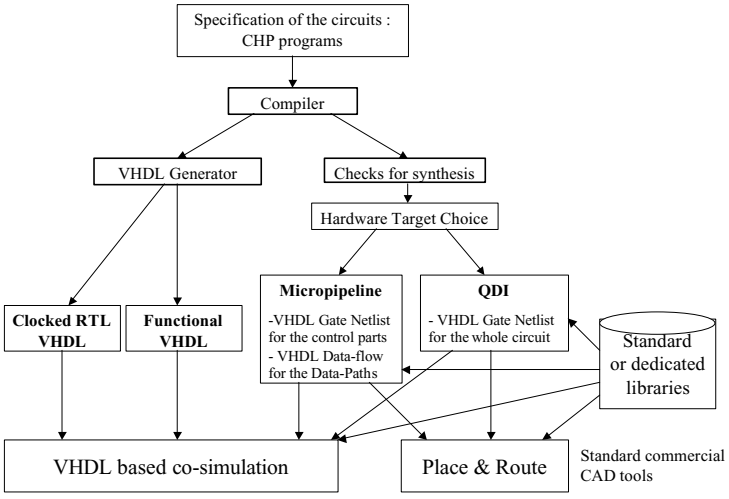

Figure 1: TAST Tool Flow

Tool/Methodology: Theseus NCL Synthesis Flow

Developer: Theseus Logic

Organisation: Theseus Logic, USA.

Summary

The NCL Synthesis flow, developed by Theseus Logic, Inc., takes advantage of NULL Convention Logic (NCL) to develop delay insensitive circuits using off the shelf synthesis tools. The NULL Convention Logic standard uses a library of threshold gates to implement multi-rail designs. . Some typical NCL threshold gates are presented in Figure 1. These cells have two states, DATA and NULL. The cells go to a DATA state when the number of inputs that go to DATA equals the threshold level for the gate. Cells in the DATA state remain in that state until all of the inputs go to NULL.

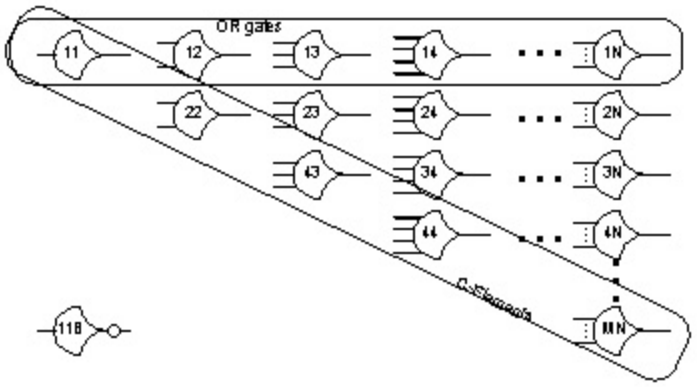

Figure 1: NCL Gate Library

NCL designs require multiple rails to represent logic values. Typically, a dual-rail structure is used, and logical functions can be built from the threshold cells.

In addition to Boolean Logic constructs, NCL designs can handle sequential structures such as Finite State Machines. The inherent storage of the threshold gates can be used to provide state storage. Therefore, a unique element is not needed to store the state.

The NCL technology, provided by Theseus Logic, Inc., provides the designer with the following advantages:

- Designs are delay insensitive. This makes them easily portable from one technology to another.

- Because there is no system clock, the EMI characteristics are considerably cleaner than a clocked design.
- If designed carefully, significant power savings can be achieved over synchronous designs.
- The hysteresis that exists in the threshold cells provides noise immunity to the circuit.
- Faults are inherently detectable due to the nature of the technology.

Strengths and Weaknesses

The NCL synthesis flow provides custom libraries and scripts to give commercially available synthesis tools the capability of mapping a design specified in VHDL to a NCL netlist. In addition to the advantages available due to the fact that the design is NCL, the synthesis flow provides the following advantages:

- Designers accustomed to synchronous circuit design can develop asynchronous circuits using a known HDL.
- Commercially available simulators can be used to verify the design.
- Simple design constraint specification during synthesis.

In addition to the synthesis support, Theseus Logic, Inc. also provides a proprietary tool, “ncl_shell”, that provides the following capabilities to the designer:

- Verification of the delay insensitive nature of the design.
- Merging of simple threshold logic cells to more complex cells.

The dual rail design methodology and the hysteresis of the NCL gates does cause the obvious disadvantages to this approach:

- The limitations of using the existing synthesis tools introduce a penalty in design area resulting in a typical 2-3X area penalty compared to the area of a Clocked Boolean Logic (CBL) design.
- The design throughput may suffer.
- A straight translation from a CBL design to NCL results in an increase in power consumption.

Some of these issues can be addressed through straightforward means. For example, custom cells that merge the threshold functions into a single cell can reduce area. Or, pipelining techniques can be used to improve throughput. Other improvements can be made that are unique to asynchronous design practices or the NCL approach. For example, using quad-rail logic instead of dual-rail logic can reduce power consumption.

With more control of the logic synthesis algorithms, results approaching manual (structured) design will be possible (e.g., Theseus has designed an 8051 microcontroller with 3X power reduction at a cost of 1.3X the transistor count of a clocked equivalent).

Application Domain

The NCL methodology is primarily a capability for synthesizing high level RTL based asynchronous designs of low-power, low EMI, medium throughput circuits. Currently focused on microcontrollers and Precision Mixed Signal designs.

Use Of Existing HDLs

The NCL approach uses existing VHDL language constructs for synthesis. There is some work underway to extend the VHDL (and Verilog) language to simplify the development of clockless circuits.

Extent of Automation

The NCL design flow allows automated synthesis from RTL code into gate-level constructs using VHDL libraries and packages. Optimal results require some manual intervention and structural coding.

Category

The NCL methodology is primarily a capability for synthesizing high level RTL based asynchronous designs of medium performance circuits. Currently focused on SmartCards and Precision Mixed Signal designs.

Design Flow and Commercial EDA Requirement

The design development flow is shown below. The following steps elaborate this process:

A. Write VHDL RTL describing design. Instantiate registration elements, and add hysteresis definitions.

B. Write pre-synthesis test bench in VHDL or Verilog.

C. Simulate the pre-synthesis netlist. Theseus Logic currently uses the “ModelTech” simulator for this; however, other simulators have been used successfully.

D. Synthesize the netlist using a custom script written for Synopsys “dc_shell”.

E. Further optimise the design using the ‘merge’ command under the Theseus Logic, Inc. tool ‘ncl_shell’.

F. Modify the pre-synthesis testbench (or write a whole new testbench) for the post-synthesis netlist. The I/O changes between the pre-synthesis netlist to the post-synthesis netlist, so the best way to handle this is to create a wrapper around the Unit Under Test. Again, the “ModelTech” simulator is used.

G. Verify that the design is free of delay sensitivities using the ‘orphan_check’ command under the Theseus Logic, Inc. tool ‘ncl_shell’.

H. Perform the Place and Route. Theseus Logic uses Silicon Ensemble.

Test Strategy

We are currently working on improved manufacturing tests. The technology is compatible with Automated Test Pattern Generation tools. In addition, the technology can have the equivalent of a SCAN register, but potentially with lower overhead than that required for Clocked Boolean Logic. For past designs, we have relied on post-synthesis functional verification. A sub-set of the functional simulation vectors were then used for manufacturing tests along with BIST to exercise memory devices.

Figure2: NCL Design Flow

Current Status of Tool

We have used NCL technology to develop small and medium size ASIC designs. This includes designs for DES/3DES encryption engines, FFTs, and microcontrollers (and related peripherals). Our most recent design was a 256 point FFT for an aerospace application.

Current Activities

Future plans for the tool suite include further development of the methodology to add pre-structured components for common structures. The support by FTL Systems for the NCL technology is expected to enable optimizations currently not possible with other commercial tool flows. In addition, Theseus Logic is currently developing an optimization technique that results in a significant decrease in area. This approach, compatible with the existing NCL RTL code, results in an approximate area reduction of 25%.

Maintainer

Theseus Logic

Availability

Theseus Logic provides design licenses, tools, training, and demo kits as standard products available to its customers. NCL technology is licensed by Theseus Logic, Inc. The synthesis flow and the ‘ncl_shell’ tool are available to licensees of the technology. The ‘ncl_shell’ tool runs under a ‘flexlm’ license.

Contact marketing@theseus.com for more information about the NCL methodology and ‘ncl_shell’.

Future Plans

Future plans for the tool include further development of the methodology to add pre-structured components for common structures. In addition, Theseus Logic is currently developing an optimization technique that results in a significant decrease in area. This approach, compatible with the existing NCL RTL code, results in an approximate area reduction of 25%.

Demonstrators

The best demonstrator to date for NCL technology is our NCL08, which is an asynchronous version of a Motorola HC08 compatible microcontroller. This design has seen insertion in sensors for containerized shipping. Theseus Logic, Inc. holds an extensive patent portfolio on NCL technology.

Contact sales@theseus.com for more information about the NCL methodology and 'ncl_shell'.

Theseus Logic, Inc. has submitted various white papers and reports on NCL technology, as well as the RTL synthesis methodology. These reports can be found at our corporate web site, *www.theseus.com*.

Tool/Methodology: Transyt

Developer: Enric Pastor, Marco A. Pea

Organisation: Universitat Politenica de Catalunya, Spain

Contributors: Jordi Cortadella (Universitat Politecnica de Catalunya, Spain) Alex Kondratyev (Theseus Logic, USA) Alex Smirnov (Universitat Politecnica de Catalunya, Spain)

Summary

Transyt is a tool for the verification of timed and untimed systems. The tool can read the description of a system and a number of safety properties to verify. Symbolic reachability based on BDDs is used to check the language containment. In case of timed systems the verification is carried out iteratively. Starting from the underlying untimed system the tool automatically inserts timing constraints until the verification is satisfied, or a timed counterexample is found.

The internal models used by Transyt are Timed Transition Systems and Lazy Transition Systems. The tool can read a Signal Transition Graph (STG) and BLIF format and translate them to the required formats. An STG is a formal specification of timing diagrams in which the events correspond to rising and falling transitions of control signals. Besides expressing causality, STGs can also specify concurrency and choice. BLIF is a format for circuit description introduced in SIS.

Transyt can be used to verify speed-independent circuits or timed circuits. However, any kind of system that can be symbolically described with a finite set of states can be analysed.

In case of timed verification Transyt provides back-annotation to the user, indicating the required timing constraints that the tool has considered during the verification.

Transyt also includes tools to visualize and analyse the set of reachable states (highlight sets of states, traces, etc.).

Strengths and Weaknesses

The tool provides fully automation during verification. The use of symbolic techniques to represent the state space allows to manage large systems.

The underlying Transition System formats are somehow cumbersome. A general high-level description language is needed.

Timed verification is very sensible to the type of system under verification. In some cases the tool is not able to select the minimum number of timing constraints to guarantee the verification. More research is needed here.

Application Domain

Verification of high-speed digital systems, both synchronous and asynchronous.

Use of existing HDLs

Netlist can be read in BLIF. Gate libraries are read in genlib format (from SIS).

Extent of automation

Complete automation of the verification process. The properties under verification should be provided by the user.

Category

Timed and untimed verification.

Design Flow and Commercial EDA Tool Requirement

Transyt can be used as a standalone tool for verification. The verification flow starts from a specification of the system's behaviour described by a Transition System. A textual format is used for such specification.

Test Strategy

No test vectors are generated.

Current Status of the Tool

Current Activities

Under development. Currently stabilizing the first public version.

Maintainer

Enric Pastor (enric@ac.upc.es)

Availability

http://research.ac.upc.es/VLSI/transyt

Future plans

Currently preparing the first public version.

Demonstrators

The tool has been used for the verification of some complex systems. In particular:

- The IPCMOS architecture: Asynchronous Interlocked Pipelined CMOS circuit architecture.
- Several delayed-reset and self-reset domino circuits.

Tool/Methodology: Veraci

Developer: Paul Cunningham

Organisation: University of Cambridge

Summary

Veraci is a tool for the formal verification of asynchronous circuits that is aimed specifically at the hardware engineer unfamiliar with formal methods. Veraci accepts as its input a circuit described in standard Verilog. Any verilog module in that circuit may also be augmented with a number of behavioural assertions and timing assumptions expressed using a novel proposition-oriented notation. Proposition-oriented behaviours extend both Petri-nets and Trace-expressions in such a way that both levels and events can be reasoned with informally. The formal semantics behind Veraci adopts a relative-time model of behaviour and its underlying verification engine is based on Binary Decision Diagrams.

Strengths and Weaknesses

Ability to reason with unusual circuits. Increased flexibility over Petri-nets and Trace-expressions. Accepts Verilog file-format directly as input.

Application Domain

Small to medium sized “hacked”-up control circuits with fragmented specifications and unusual timing-assumptions.

Category

Formal verification. Proposition-oriented behaviours.

Design Flow

Any design flow where circuits are described in the Verilog HDL.

Current Status of Tool

Current Activities

Under development.

Maintainer

Paul Cunningham <pac22@cl.cam.ac.uk>.

Future Plans

Verification of liveness/progress properties. Hierarchical abstraction

Demonstrators

There are currently no formal publications relating to Veraci.

Tool/Methodology: VeriMap

Developer: Danil Sokolov

Organisation: School of Electrical, Electronic and Computer Engineering, University of Newcastle, Newcastle upon Tyne, UK

Summary

The VeriMap design kit converts single-rail RTL netlists into dual-rail circuits which are resistant to Differential Power Analysis (DPA) attacks. Verimap design kit successfully interfaces to the Cadence CAD tools. It takes as input a structural Verilog netlist file, created by Cadence Ambit (or another logic synthesis tool), and converts it into dual-rail netlist. The resulting netlist can then be processed by Cadence or other EDA tools. All Design For Testability (DFT) features incorporated at the logic synthesis stage are preserved.

The structure of our Verimap design kit is displayed in figure 1. The main parts are the tool itself and two libraries. The library of gate prototypes contains the description of gates used in the input netlist. It facilitates the structural analysis of the input netlist. The library of transformation rules defines: complementary gates needed for construction of the dual-rail logic, the polarity of gate inputs and outputs and specifies if the corresponding dual-rail gate requires completion signal (for asynchronous design only) and if it inverts the spacer. If a predefined dual-rail implementation of a gate is found in the library the tool uses it, otherwise an implementation is built automatically using the rules.

The main function of the tool is conversion of a single-rail RTL netlist into a dual-rail netlist of either of two architectures: self-timed and clocked. It is done in four stages. Firstly, a single-rail circuit is converted into positive logic dual-rail. Secondly, the positive dual-rail gates are replaced by negative dual-rail gates and the spacer polarity inverters are inserted. This is the negative gate optimisation [1]. Then, the completion signal is generated (asynchronous design only). Finally, a wrapper module connecting the dual-rail circuit to the single-rail environment is added (optional).

Using the standard dual-rail protocol with a single spacer has certain balancing problems due to the asymmetry between logic gates within a dual-rail gate. We addressed and solved these problems in [2,3] by using a new protocol with two spacers alternating in time leading to all gates switching within every clock cycle. The tool supports the alternating spacer protocol.

Apart from generating netlists, Verimap tool reports statistics for the original and resultant circuits: estimated area of combinational logic and flip-flops, number of negative gates and transistors, number of wires.

The tool also generates a behavioural Verilog file assisting the power analysis of the original and resultant circuits. Being included into simulation testbench these Verilog counts the number of switching events in each wire of the circuits.

Strengths and Weaknesses

Strengths.

The tool preserves the industry design flow and interfaces to standard CAD tools (Cadence).

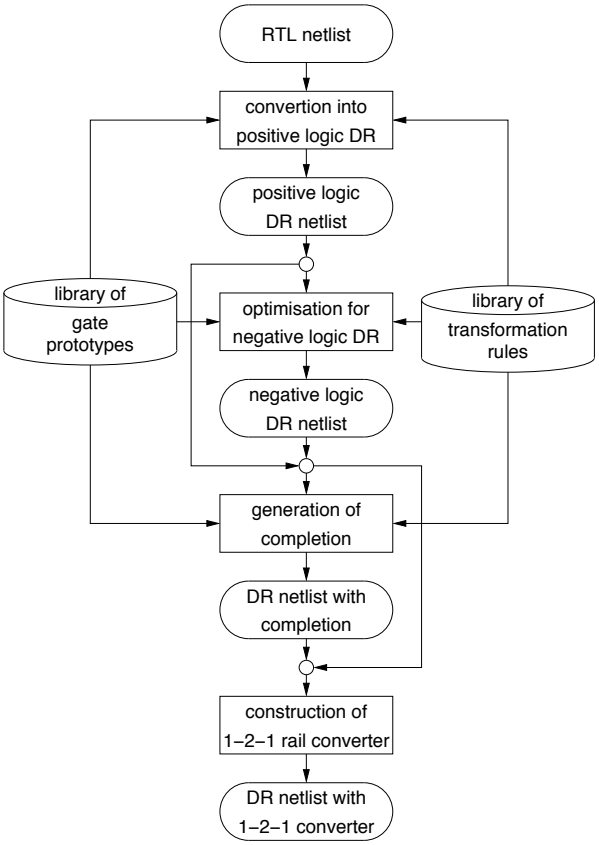

Figure 1: Verimap Design Kit

- Generated circuits are hazard free and resistant to DPA attacks.

- Support for two architectures: self-timed dual-rail or clocked dual-rail.

- Generated circuits preserve all DFT features incorporated at the logic synthesis stage.

- Support for negative logic optimisation to reduce the size of the circuit and shorten the critical path.

- Support for the alternating spacer protocol to resist DPA attacks by making the power consumption data-independent.

Weaknesses:

- For each technology the library of transformation rules has to be created manually.

- Clock gating and data guarding are not implemented yet.

Application Domain

Secure systems resistant to DPA attacks.

Use of Existing HDLs

The tool takes as an input a structural Verilog netlist of a single-rail circuit and converts it into a Verilog netlist of a dual-rail circuit.

Extent of Automation

The tool requires a library of transformation rules, which should be created for each foundries' technology. It is done once for each technology and can be reused for different circuits. This is the only part which is not automated yet. The rest of the conversion process if automatic, though flexible and can be adjusted to the needs of the designer.

Category

Synthesis - Direct mapping

Design Flow and Commercial EDA Requirement

The Verimap design kit maintains the industry design flow. It successfully interfaces to the Cadence CAD tools. It takes as input a structural Verilog netlist file, created by Cadence Ambit (or another logic synthesis tool), and converts it into dual-rail netlist. The resulting netlist can then be processed by Cadence or other EDA tools. Furthermore, all Design For Testability (DFT) features incorporated at the logic synthesis stage are preserved.

Current Status

Current Activities

The tool is currently in testing stage.

Maintainer

Danil Sokolov (danil.sokolov@ncl.ac.uk)

Availability

http://async.org.uk/

Future Plans

- To automate (or simplify) the process of transformation rules library generation.
- To implement clock gating and data guarding.
- To fabricate a demonstrator and try an actual DPA attack.

Demonstrators

N/A

References

[1] A.Bystrov, D.Sokolov, A.Yakovlev, A.Koelmans: "Balancing Power Signature in Secure Systems". 14th UK Asynchronous Forum, Newcastle, June 2003.

[2] D.Sokolov, J.Murphy, A.Bystrov, A.Yakovlev: "Improving the security of dual-rail circuits". Proc. CHES'04, Springer LNCS, Boston, August 2004.

[3] D.Sokolov, J.Murphy, A.Bystrov, A.Yakovlev: "Improving the security of dual-rail circuits", Technical report, Microelectronic System Design Group, School of EECE,

University of Newcastle upon Tyne, April 2004, http://www.staff.ncl.ac.uk/i.g.clark/async/tech-reports/NCL-EECE-MSD-TR-2004-101.pdf

Tool/Methodology: VerySAT

Developer: Victor Khomenko

Organisation School of Computing Science, University of Newcastle upon Tyne

Summary

VerySAT is a SAT based model checker. It can formally verify various safety properties (deadlock-freeness, mutex, reachability, etc.), detect coding (CSC and USC) conflicts in STGs and derive a complex-gate implementation of a circuit. VerySAT employs finite complete prefixes of Petri net unfoldings (e.g. those produced by the Punf tool [3]).

Strengths and Weaknesses

Strengths. memory efficient and very fast, derives a very good complex-gate implementation of a circuit.

Weakness. the tool is at the alpha-stage.

Application domain

Synthesis of self-timed circuits and formal verification.

Use of existing HDLs

None (works on finite prefixes in the `.mci' format, e.g. those generated by Punf from STGs).

Extent of automation

Fully Automated

Category

Synthesis and verification =>STG/Petri net.

Design Flow and Commercial EDA Tool Requirement

VerySAT is intended as a powerful model checking and synthesis engine to be used by other applications.Currently, prefixes produced by Punf can be used by VerySAT, Clp and ConfRes[4,6,8,9.10] (described in this Report) for detection and resolution of coding (CSC and USC) conflicts in STGs and for deriving a complex-gate implementation of a circuit. Thus these tools comprise a complete framework for complex-gate synthesis of speed-independent circuits from STGs.

Test strategy

VerySAT can formally verify various safety properties, such as deadlock-freeness, mutex, reachability, etc.

Current status

VerySAT is at the alpha-stage.

Maintainer

Victor Khomenko Victor.Khomenko@ncl.ac.uk

Availability

Will eventually be made available for research purposes from.

http://www.cs.ncl.ac.uk/people/victor.khomenko/home.formal/tools/tools.html

Future plans

We plan to create a full design cycle for self-timed circuits based on STG unfoldings and not involving building the state space at any stage. VerySAT is to replace Clp in near future as its performance is much better.

References

[1] E.Best and B.Grahlmann: *"PEP: Documentation and User Guide, Version 1.4. Manual"* (1995).

[2] K.Heljanko, V.Khomenko, and M.Koutny: *"Parallelisation of the Petri Net Unfolding Algorithm"*. TACAS'2002, LNCS 2280 (2002) 371-385.

[3] V.Khomenko: *"Punf: Documentation and User Guide, Version 6.01. Manual"* (2002).

[4] V.Khomenko: Clp: *"Documentation and User Guide. Version 3.01beta. Manual"* (2002).

[5] V.Khomenko and M.Koutny: *"Towards An Efficient Algorithm for Unfolding Petri Nets"*. CONCUR'2001, LNCS 2154 (2001) 366-380.

[6] V.Khomenko, M.Koutny, and A.Yakovlev: *"Detecting State Coding Conflicts in STGs Using Integer Programming"*. DATE'2002, IEEE Comp. Soc. Press (2002) 338-345.

[7] V.Khomenko and M.Koutny: *"Branching Processes of High-Level Petri Nets"*. TACAS'2003, LNCS (2003)

[8] V.Khomenko, M.Koutny and A.Yakovlev: *"Detecting State Coding Conflicts in STG Unfoldings Using SAT"* Fundamentae Informatica, Sepecial Issue on Best Papers from ICACSD'2003 (to appear).

[9] V.Khomenko, M.Koutny and A.Yakovlev: *"Logic Synthesis Avoiding State Space Explosion"* ICACSD '04. (2004 to appear).

[10] A.Madalinski, A.Bystrov, V.Khomenko, and A.Yakovlev: *"Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design"*. IEE Proceedings: Computers & Digital Techniques: Special Issue on Best Papers from DATE'2003.(2003)

Tool/Methodology: VSTGL

Developer: Hans P. Palb and Sune Frankild

Organisation: Technical University of Denmark, DTU.

Summary

Visual STG Lab is a graphics tool for capturing and simulating STGs. Asynchronous control circuits are often specified using signal Transition Graphs (STG) and synthesized using tools like Petrify whose input format is a textual description of the STG. Visual STG Lab (VSTGL) is a graphics editor and test environment for creating STGs, and it can be used as a front-end to Petrify. Compared with the normal Petrify design flow, VSTGL offers several advantages resulting in a faster and less error prone design flow:

- VSTGL allows graphical entry of the STG and checking of structural properties.
- VSTGL allows the designer to simulate the STG before synthesis (by placing tokens and firing transitions).
- VSTGL outputs the STG for documentation (.eps file) exactly as the designer entered it. This sounds trivial, but as the designer normally expresses key ideas about the design in the topological structure of the STG this is important. (Petrify uses the program *dot*, and normally it produces drawings that bear little resemblance with what the designer had in mind).
- When the designer is satisfied with the STG, an input file for Petrify can be generated, or Petrify can be invoked from within STG.

Application Domain

- Synthesis
- STG/Petri net
- Notation to notation conversion

Current Status of Tool

Current Activity

VSTGL was originally developed in a special topic course at the Technical University of Denmark (DTU). The authors have since continued to improve and extend the tool. Students in a course on asynchronous circuit design at DTU have used the current version in the fall 1999 and again in the fall 2001 and it proved to be a stable and very useful tool for designing asynchronous control circuits

Availability

VSTGL is a public domain tool. It runs under Linux, and it may be downloaded from: *http://sourceforge.net/projects/vstgl/* and *http://vstgl.sourceforge.net/* Check which page is being updated.

Tool/Methodology: Weaver/Gate Transfer Level (GTL) synthesis

Developers: Alexandre Smirnov, Alexander Taubin

Organisation: Department of Electrical and Computer Engineering, Boston University

Summary

Weaver is an asynchronous EDA flow targeting synthesis of quasi-delay-insensitive (QDI) fine-grain pipelined circuits from high-level behavioral specification. Weaver is based on asynchronous Gate Transfer Level (GTL) synthesis methodology assuming that gates (or small portions of logic) implemented as pipeline stages that interact with each other using handshake protocols.

Weaver synthesis flow utilises commercial synthesis engine(s) to

- provide quality support for the behavior specification formats;

- provide a familiar interface (for synchronous RTL designers);

- provide a smooth interface to simulation and post-synthesis tools;

- enable synthesis of large (industry sized) designs in acceptable time;

- reduce the time and resources needed for its development.

Delay insensitivity is ensured by the handshake protocol (various protocols can be used) and by implementing the whole pipeline stages as library cells so that in-stage timing assumptions are met by the cell design and no longer depend on placement and routing.

The current version of the tool (still a prerelease 0.81) utilises the Synopsys Design Compiler (DC-Ultra) as a host synthesis engine.

Strengths and Weaknesses

Strengths:

- fast (the synthesis time starting from the behavior specification is approximately 2.5x the time of Synopsys-DC run for the same design);

- fully supports standard synthesizable subset of Verilog, VHDL as well as other languages supported by Synopsys DC-Ultra as input specifications;

- supports standard intermediate and output netlist formats for easy integration with simulation and P&R tools;

- fully automated synthesis of fine-grain pipelined implementation;

- support for various pipelining styles;

Weaknesses:

- no dedicated library with optimal area/performance ratio developed yet (PCHB library developed at USC can be used if the resetting capability is provided for the cells, but that library is optimized for performance only).

Application domain

Synthesis of quasi-delay-insensitive (QDI) fine-grain pipelined [asynchronous] circuits.

Use of existing HDLs

Synthesizable HDL subset.

Extent of automation

Fully automatic; customization possible.

Category

Synthesis: HDL behavior specification to a P&R ready netlist (mapped to a dedicated ASIC library comprising dual-rail QDI stage-like implementations for the basic logic functions).

Design flow

Input: RTL behavioral/structural/mixed possibly hierarchical synchronous style high-level specification

Step1: RTL synthesis by Synopsys DC-Ultra

Step2: Weaving: dual-rail expansion and synthesis of handshake circuitry for the RTL implementation of the specified behavior by the Weaver Engine;

Step3: mapping the Weaved netlist into a GTL library comprising dual-rail implementation of logic functions, completion detection and handshake implementation;

Output: netlist of GTL library cells in any format supported by Synopsys DC-Ultra

The flow is shown in figure 1. The srGTL stands for a virtual single-rail library functionally equivalent to the GTL library.

Commercial EDA Requirement

Requires Synopsys DC-Ultra and Synopsys ACS for synthesis, simulation is supported for Model Technology ModelSim (distributed by Mentor Graphics).

Current status of the Tool

Prerelease. MCNC benchmarks: multilevel combinational benchmarks – no problems, problems with some of the FSM benchmarks. The first release will be made as soon as the synthesis produces valid implementations for all MCNC HDL benchmarks including the HLSynth’95 set.

Maintainer

Alexandre Smirnov (alexbs@bu.edu).

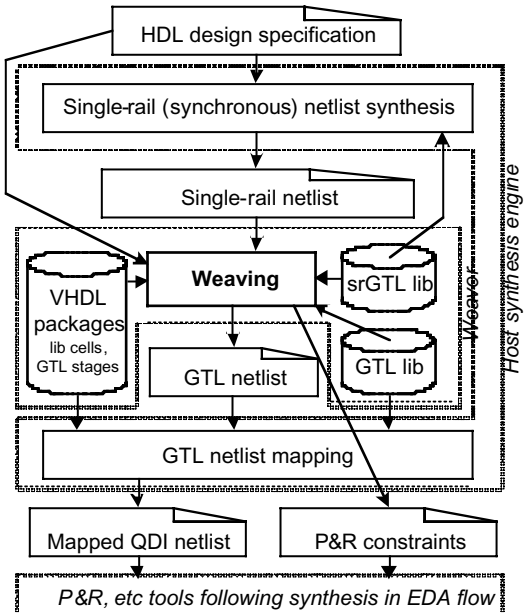

Figure 1: Weaver Design Flow

Availability

Available for research purposes from http://async.bu.edu/weaver/

Future plans

A full release of the tool. Further plans: dedicated library development, increasing efficiency (mostly from the performance point of view) of the synthesized circuits, providing support for FPGA design (using Xilinx FPGAs first).

References

[1] Smirnov A., Taubin A., and Karpovsky M. "*Automated Pipelining in ASIC Synthesis Methodology: Gate Transfer Level.*" in IWLS 2004 Thirteenth International Workshop on Logic and Synthesis. June 2-4, 2004. Temecula, California, USA.

[2] Smirnov A., Taubin A., Karpovsky M. Rozenblyum L. "*Gate and Transfer Level Synthesis as an Automated Approach to Fine-Grain Pipelining.*" Workshop on Token Based Computing (ToBaCo). June 22, 2004. Bologna, Italy.

[3] Smirnov A., Taubin A. "*Weaver: installation guide.*" Distributed at *http://async.bu.edu/weaver/*

[4] Smirnov A., Taubin A. "*Weaver: user guide*". Distributed at *http://async.bu.edu/weaver/*

[5] Smirnov A., Taubin A. "*Weaver: library developer guide*". Distributed at *http://async.bu.edu/weaver/*

Tool/Methodology: XDI

Developer: Tom. Verhoeff

Organisation: Eindhoven University of Technology

Summary

The XDI Model (eXtended model for Delay-Insensitive systems) is a theoretical framework to argue about the specification and implementation of delay-insensitive systems. It includes a refinement (satisfaction) relation and a composition operator. It is state based, where the extension addresses some form of progress. For details see

There is a tool available with an e-mail interface to have XDI specifications analysed and the resulting report sent back. The tool uses the AND/IF notation, which a general notational framework for finite-state specifications. See *http://edis.win.tue.nl/and-if/mailproc.html*

Design Flow and Commercial EDA Tool Requirement

The theory is used by Willem Mallon to help decompose specifications into synthesizable blocks through his tools (in particular, Ludwig). The analysis tool is used to get a better understanding of specifications before an attempt to implement them is being made. No commercial EDA tools are used

Test Strategy.

Not relevant.

Current Status of Tool

Current Activities

Maintainer

Not maintained

Availability

See *http://edis.win.tue.nl/and-if/mailproc.html*

Future Plans

All very vague at the moment.

References

A Theory of Delay-Insensitive Systems, Tom Verhoeff, Dissertation, Eindhoven University of Technology, Department of Computing Science, 1994.

Analysing Specifications for Delay-Insensitive Circuits, Tom Verhoeff, Appeared in Proc. of the Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems, IEEE Press, 15 pages (incl. abstract, figures, references), Async’98 in San Diego, 30 March - 2 April, 1998.

Analysis and Applications of the XDI Model. Willem C. Mallon, Jan Tijmen Udding, Tom Verhoeff. Appeared in Proc. of the Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems, April 1999, IEEE Press, pp.231-242. (Async'99, 19-21 April, Barcelona, Spain.)

On the web, there is an Encyclopedia of Delay-Insensitive Systems (EDIS). See *http://edis.win.tue.nl/edis.html*

Testing Asynchronous Circuits

Asynchronous circuits exhibit several properties that make them harder to test for production faults than their synchronous counterparts:

- The lack of a global synchronisation clock: this decreases the level of controllability over the states of the circuit, making single stepping through states difficult, as the ability to stop the circuit in a given state becomes more complicated.
- Large numbers of state holding elements, meaning generating tests is harder, and Design For Test techniques require greater overhead.
- Detection of Hazards and races is more critical in an asynchronous environment.
- Logic redundancy is introduced in order to eliminate hazards, this makes stuck at faults in redundant parts of the circuit impossible to test.

However, because asynchronous circuits use handshaking techniques, in the presence of faults handshakes become unfulfilled and the circuit deadlocks, allowing the fault to be detected easily. This property is known as self-checking or self-diagnostic. By selecting the correct fault model several classes of asynchronous circuit can be made to exhibit this property. Testing can also be made easy by several design for testability techniques which allow for more observability of a finished design.

The following is a selected bibliography of asynchronous testing references. Much of the work done in this field concerns adapting existing testing techniques to an asynchronous paradigm. Ref. [16] provides a general introduction to fault models, and design for testability, Self-diagnostic properties are discussed in [3],[4],[10], design for testability is discussed in [15], [17], [27],[35], with specific design styles being presented in [19], [46].

[1] S.Banerjee, S.T. Chakradhar, R.L.Roy. “Synchronous test generation model for asynchronous circuits”, *Proceedings of the International Conference on VLSI Design*, Bangalore, January 1996.

[2] W.J.Bainbridge, L.A.Plana, S.B.Furber. “The Design and Test of a Smartcard Chip Using a CHAIN Self-timed Network-on-Chip. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition Designers' Forum*, April 2004.

[3] P.A. Beerel, T.H.Y.Meng, “Semi-modularity and testability of speed-independent circuits”, *Integration, The VLSI Journal*, 13(3), Sept., 1992, pp. 301-322.

[4] P.A.Beerel and T.H.Y.Meng, “Testability of asynchronous timed control circuits”, *Proceedings Design Automation Conference*, pp. 446-451, June 1991.

[5] C.H.Kees van Berkel, R.Burgess, J.Kessels, M.Roncken, F.Schalij, A.Peeters, “Asynchronous circuits for low power: A DCC error corrector”, *IEEE Design & Test of Computers*, 11(2), pp.22-32, 1994.

[6] J.A.Brzozowski, K.Raahemifar, “Testing C-elements is not elementary”, *Proceedings 2nd Working Conference on Asynchronous Design Methodologies*, South Bank University, London, May 30-31, pp. 150-159, 1995.

[7] J.A.Brzozowski, C.-J.Seger, “A Unified Framework for race analysis of asynchronous networks”, *Journal of the ACM*, 36(1), pp. 20-45, 1989.

[8] T.J.Chaney, C.E.Molnar, “Anomalous behavior of synchronizer and arbiter circuits”, *IEEE Transactions on Computers*, C-22(4), pp. 421-422, April 1973.

[9] G.R.Carson, G.Borriello, "A testable CMOS asynchronous counter", *IEEE Journal of Solid State Circuits* 5(4), pp. 952-960, 1990.
[10] I.David, R.Ginosar, M.Yoeli, "Self-timed is self-diagnostic", *Technical Report TR-UT-84112*, Department of Computer Science, University of Utah, 1990.
[11] A. Efthymiou, C. Sotiriou, D.A. Edwards. "Automatic Scan Insertion and Pattern Generation for Asynchronous Circuits" *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, Feb. 2004.
[12]
[13] F. Gurkaynak, T. Villiger, S. Oetiker, N. Felber, H Kaeslin, W. Fichtner, "A Functional Test Methodology for Globally-Asychronous Locally-Synchronous Systems",*Proceedings Eighth International Symposium on Asynchronous Circuits and Systems,* April 2002.
[14] D.S.Ha, S.M.Reddy, "One testable self-timed logic circuits*14dings International Conference on Computer Design*, pp. 296-301, 1984.
[15] P.Hazewindus, "Testing delay-insensitive circuits", Ph.D.thesis, California Institute of Technology, 1992.
[16] H.Hulgaard, S.M.Burns, G.Borriello, "Testing asynchronous circuits: A survey", *Integration, The VLSI Journal*, 19(3), pp. 111-13, 1995.
[17] K.Keutzer, L.Lavagno, A.Sangiovanni-Vincentelli, "Synthesis for testability techniques for asynchronous circuits", *International Conference on Computer-Aided Designs*, pp. 326-329, 1991
[18] K.Keutzer, L.Lavagno, A.Sangiovanni-Vincentelli, "Synthesis for testability techniques for asynchronous circuits", *IEEE Transactions on Computer-Aided Design*, 11(1), pp. 87-101, 1995.
[19] A.Khoche, E.Brunvand, "Testing micropipelines", *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems (Async94)*, Utah, pp. 239-246, 1994.
[20] A.Khoche, E.Brunvand, "A partial scan methodology for testing self-timed circuits", *Proceedings 13th IEEE VLSI Test Symposium*, Princeton, New Jersey, USA, pp. 283-289, May 1995.
[21] A.Khoche and E.Brunvand. "Testing self-timed circuits using partial scan", *Proceedings of the 2nd Working Conference on Asynchronous Design Methodologies*, pp. 160-169, London, May 1995.
[22] A.Khoche, E.Brunvand, "Critical hazard free test generation for asynchronous circuits", *Proceedings IEEE VLSI Test Symposium*, pp. 203-208, 1997.
[23] M.Kishinevsky, A.Kondratyev, L.Lavagno, A.Saldanha, A.Taubin, "Partial scan delay fault testing of asynchronous circuits", *Proceedings ICCAD*, pp. 728-735, 1997.
[24] A. Kondratyev, L. Sorenson, A. Streich, "Testing of Asynchronous Designs by "Inappropriate" Means: Synchronous Approach", *Proceedings Eighth International Symposium on Asynchronous Circuits and Systems*, April 2002.
[25] L.Lavagno, A.Sangiovanni-Vincentelli, "Algorithms for synthesis and testing of asynchronous circuits", *Kluwer Academic Publishers*, 1993.
[26] L.Lavagno, "Synthesis and testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs", *Phd Thesis*, University of California at Berkeley, 1992.
[27] L.Lavagno, M.Kishinevsky, and A.Lioy, "Testing redundant asynchronous circuits", Proceedings European Design Automaton Conference (EURO-DAC), *IEEE Computer Society Press*, 1994.

[28] H.K.Lee and D.S.Ha, “On the generation of test patterns for combinational circuits”, *Technical Report No. 12_93*, Dept.of Electrical Eng, Virginia Polytechnic Institute and State University, 1993.
[29] T.Li, “Design for VLSI asynchronous circuits for testability”, *International Journal of Electronics*, 64(6), pp. 859-868, 1988.
[30] A.J.Martin, P.J.Hazewindus, “Testing Delay Insensitive Circuits”, *Proceedings Advanced Research in VLSI*, MIT Press, pp. 118-132, 1991.
[31] S.M.Nowick, N.K.Jha, F.C. Cheung, “Synthesis of asynchronous circuits for stuck-at and robust path delay fault testability”, *IEEE Transactions Computer-Aided Design*, vol. 16, pp. 1514-1521, Dec. 1997.
[32] R. Negulescu, “General Testers for Asynchronous Circuits”,*Proceedings International Symposium Advanced Research in Asynchronous Circuits and Systems*, 2004.
[33] S.Pagey, G.Venkatesh, S.Sherlekar, “Issues in fault modelling and testing of micropipelines”, *First Asian Test Symposium*, Hiroshima, Japan, Nov. 1992.
[34] O.A.Petlin, “Random testing of asynchronous VLSI circuits”, *MSc Thesis*, University of Manchester, 1994.
[35] O.A.Petlin, “Design for testability of asynchronous VLSI circuits”, *Phd Thesis*, University of Manchester, 1996.
[36] O.A.Petlin, S.B.Furber, “Designing C-elements for testability”, *Technical Report UMCS-95-10-2*, Department of Computer Science, University of Manchester, UK, 1995.
[37] O.A.Petlin, S.B.Furber, A.M.Romankevich, V.V.Groll, “Designing asynchronous sequential circuits for random pattern testability”, *IEE Proceedings Comput. Digit. Tech.*, 142(4), July 1995.
[38] O.A.Petlin, S.B.Furber, “Scan testing of asynchronous sequential circuits”, *Proceedings 5th Great Lakes Symposium on VLSI*, New York, pp. 224-229, March 1995.
[39] O.A.Petlin, S.B.Furber, “Scan testing of micropipelines”, *Proceedings 13th IEEE VLSI Test Symposium*, Princeton, New Jersey, USA, pp. 296-301, May 1995.
[40] O.A.Petlin, S.B.Furber, “Built in scan testing of micropipelines”, *Proceedings International Symposium Advanced Research in Asynchronous Circuits and Systems*, Eindhoven, pp. 22-29, 1997.
[41] O.A.Petlin, C.Farnsworth, S.B.Furber, “Design for testability of an asynchronous adder”, *Proceedings of IEE Colloquium on Design and Test of Asynchronous Systems*, London, UK, pp. 5/1- 5/9, 1996.
[42] G.R.Putzolu, J.P.Roth, “A heuristic algorithm for the testing of asynchronous circuits”, *IEEE Transactions on Computers*, C-20(6), pp. 639-647, June 1971.
[43] O.Roig, J.Cortadella, E.Pastor, “Automatic Generation of synchronous test patterns for asynchronous circuits”, *Proceedings 3th Design Automaton Conference*, June 1997.
[44] O Roig, “Formal Verification and Testing of Asynchronous Circuits, *Phd Thesis*, Universitat Politecnica de Catalunya”, May 1997.
[45] M.Roncken, “Defect-oriented testability for asynchronous ICs”, *Proceedings of the IEEE*, 87(2), pp. 363-375, February 1999.
[46] M.Roncken, “Partial scan test for asynchronous circuits illustrated on a DCC error corrector”, Proceedings *International Symposium on Advanced Research in Asynchronous Circuits and Systems (Async94)*, pp. 247-256, Nov. 1994.
[47] M.Roncken, R.Saeijs, “Linear test times for delay-insensitive circuits: a compilation strategy”, *IFIP WG 10.5 Working Conference on Asynchronous Design Methodologies*, Editors S.Furber, M.Edwards, Manchester, pp. 13-27, 1993.

[48] M.Roncken, K.Stevens, R.Pendurkar, S.Rotem, and P.Chaudhuri, “CA-BIST for asynchronous circuits: A case study on the RAPPID asynchronous instruction length decoder”, *Proceedings International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pp. 62-72, 2000.
[49] M.Roncken. “Defect-Oriented Testability for Asynchronous ICs”, *Proceedings of the IEEE*, 87(2), pp. 363-375, Feb. 1999.
[50] M.Roncken, E.Bruls. “Test Quality of Asynchronous Circuits: A Defect-Oriented Evaluation”, *Proceedings International Test Conference*, pp. 205-214, 1996.
[51] M.Roncken, E.Aarts, W.Verhaegh, “Optimal Scan for pipelined testing: An asynchronous foundation”, *Proceedings International Test Conference*, pp. 215-224, 1996.
[52] J.Saxena, D.K.Pradhan, “Design for testability of asynchronous sequential circuits”, *Proceedings International Conference on Computer Design*, pp. 518-522, 1983.
[53] A.K.Susskind, “A technique for making asynchronous sequential circuits readily testable”, *Proceedings International Test Conference*, pp.842-846, 1984.
[54] V.Schober, T.Kiel, “An asynchronous scan path concept for micropipelines using the bundled data convention”, *Proceedings International Test Conference*, pp.225-231, 1996.
[55] M.D.Shieh, C.L.Wey, P.D.Fisher, “A scan design for asynchronous sequential logic circuits using SR-latches”, *Proceedings Midwest Symposium Circuits and Systems*, pp.1300-1303, 1993.
[56] F. te Beest, A Peeters, K. Van Berkel, H. Kerkhoff. Automatic Structural Test Generation for Asynchronous Circuits, *Proceedings IEEE/ProRISC Symposium on Circuits, Systems and Signal Processing*, Nov. 2001.
[57] C.L.Wey, M.D.Shieh, P.D.Fisher, “ASCLScan: a scan design for asynchronous sequential logic circuits”, *Proceedings IEEE International Conference on Computer-Aided Design*, pp.159-162, 1993.
[58] K. van Berkel, A. Peeters, F. te Beest, “Adding Synchronous and LSSD Modes to Asynchronous Circuits”, *Proceedings Eighth International Symposium on Asynchronous Circuits and Systems*, April 2002.

Appendix A: Identified Tools & Methodologies

This appendix lists methodologies, tools and approaches identified which have some bearing on the design, implementation, verification or testing of asynchronous circuits. Note that the organisation refers to the institution in which the tool/methodology was originally developed, not necessarily where development of the tool is current. All developers on this listed were requested to respond to the email questionnaire.

Methodologies

Name:	Burst Mode Machine
Organisation:	Columbia University, New York, USA
Contact	nowick@cs.columbia.edu
Comment	

Name:	Communicating Hardware Processes
Organisation:	University of Leeds, UK
Contact	graham@comp.leeds.ac.uk
Comment	

Name:	Communicating Hardware Processes
Organisation:	Caltech
Contact	alain@cs.caltech.edu
Comment	

Name:	Delay Insensitive Algebra
Organisation:	South Bank University
Contact	Mark.Josephs@sbu.ac.uk
Comment	

Name:	Handshake Circuits Organisation
Organisation:	Philips research Laboratories
Contact	Ad.Peeters@philips.com
Comment	

Name:	Micropipelines/GASP
Organisation:	Sun microsystems
Contact	jo.ebergen@eng.sun.com
Comment	

Name:	Null Convention Logic
Organisation:	Theseus Logic
Contact	dlamb@theseus.com
Comment	

Name:	Petri Nets and Signal Transition Graphs
Organisation:	University of Newcastle
Contact	Alex.Yakovlev@ncl.ac.uk
Comment	

Name:	eXtended Delay Insensitive Model
Organisation:	TUE
Contact	wstomv@win.tue.nl
Comment	

Burst-mode synthesizers

Name:	MEAT
Organisation:	University of Calgary
Author	Ken Stevens
Contact	kstevens@ichips.intel.com
Availability	Available – LISP source
Located	ftp://kdstevens.com/pub/stevens/meat.tar.gz
Dated	21/06/00
Maintenance	Unknown
Comment	Author left Calgary, moved to Intel

Name:	MINIMALIST
Organisation:	Columbia University, NY
Author	Robert Fuhrer
Contact	rmf@cs.columbia.edu
Availability	Available
Located	http://www.cs.columbia.edu/~rmf/MINIMALIST-get.html
Dated	26/08/99
Maintenance	Maintained
Comment	Contact Steve Nowick for further details

Name:	3D
Organisation:	University of California, San Diego
Author	Ken Y. Yun
Contact	kyy@ucsd.edu
Availability	Available – C source & Sun/HP binaries
Located	http://paradise.ucsd.edu/3D/index.htm
Date	13/07/99
Maintenance	Maintained
Comment	Author resident at UC San Diego

STG/Petri-Net Synthesisers

Name:	ASSASSIN
Organisation:	IMEC
Author	Chantel Ykman
Contact	ykman@imec.be
Availability	Available
Located	ftp://ftp.imec.be/pub/vsdm/SW-distrib/assassin/
Date	22/08/95
Maintenance	Unknown
Comment	No recent information can be found. try also: assassin@imec.be

Name:	FORCAGE
Organisation:	University. of Aizu
Author	Michael Kishinevsky
Contact	mkishine@ichips.intel.com
Availability	Available
Located	ftp://ftp.it.dtu.dk/pub/forcage/forcage3.zip
Date	18/01/94
Maintenance	Unknown
Comment	Kishinevsky, has left Aizu, now at Intel. Old (DOS) version available from DTU.

Name:	Petrify
Organisation:	Universitat Politenica de Catalunya, Spain
Author	Jordi Cortadella
Contact	Jordi Cortadella (jordic@lsi.upc.es)
Availability	Available
Located	http://www.lsi.upc.es/~jordic/petrify
Date	
Maintenance	Maintained
Comment	

Name:	SIS
Organisation:	Berkley
Author	SIS Team (Luciano Lavagno, Cho Moon, Paul Stephan)
Contact	sis@ic.eecs.berkeley.edu
Availability	Available
Located	ftp://ic.eecs.berkeley.edu/pub/Sis
Date	22/03/00 (Redhat Version)
Maintenance	No longer actively maintained
Comment	Luciano Lavagno's email: lavagno@polito.it

Name:	SYN
Organisation:	Stanford
Author	Peter Beerel
Contact	pabeerel@eiger.usc.edu
Availability	Not Available
Located	
Date	
Maintenance	Unknown
Comment	Beerel has left Stanford and is now at USC.

Silicon compilers (& Macromodular Synthesis)

Name:	Balsa
Organisation:	Amulet Group, University of Manchester
Author	Andrew Bardsely
Contact	balsa@cs.man.ac.uk
Availability	Available
Located	ftp://ftp.cs.man.ac.uk/pub/amulet/balsa
Date	11/09/00
Maintenance	Maintained
Comment	

Name:	Tangram
Organisation:	Philips
Author	Tangram team
Contact	Ad.Peeters@philips.com
Availability	Not Available
Located	
Date	
Maintenance	Maintained
Comment	Used internally within Philips – not for general use

Name:	TAST
Organisation:	TIMA-CMP
Author	Marc Renaudin
Contact	marc.renaudin@imag.fr
Availability	Not available yet
Located	Tima/CIS
Date	
Maintenance	Maintained - under development
Comment	Tool is planned to be fully operational for the Summer School on Asynchronous Circuit Design to be held in Grenoble in July 2002.

Timing Synthesisers

Name:	ATACS
Organisation:	University of Utah
Author	Chris J.Myers
Contact	myers@ee.utah.edu
Availability	Not Available
Located	
Date	
Maintenance	Maintenance unknown
Comment	http://www.async.elen.utah.edu/tools.html

Logic Minimisers

Name:	HFMIN
Organisation:	Columbia University, NY
Author	Robert Fuhrer
Contact	rmf@cs.columbia.edu
Availability	Available
Located	http://www.cs.columbia.edu/~rmf/MINIMALIST-get.html
Date	11/06/99
Maintenance	Not maintained
Comment	Available with Minimalist and 3D, from UCSD

Name:	IMPYMIN
Organisation:	Columbia University, NY
Author	Michael Theobald
Contact	theobald@cs.columbia.edu
Availability	Available
Located	http://www.cs.columbia.edu/~rmf/MINIMALIST-get.html
Date	
Maintenance	Not known
Comment	Available as part of Minimalist only

Name:	ESPRESSO-HF
Organisation:	Columbia University, NY
Author	Michael Theobald
Contact	theobald@cs.columbia.edu
Availability	Available
Located	http://vstgl.sourceforge.net/download.html
Date	19/01/00
Maintenance	Maintained
Comment	Available as part of Minimalist only

Design Entry

Name:	VSTGL
Organisation:	DTU
Author	Sune Frankild
Contact	sune.frankild@get2net.dk
Availability	Available
Located	http://vstgl.sourceforge.net/download.html
Date	19/01/00
Maintenance	Maintained
Comment	

Extensions to Existing HDLs

Name:	VHDL++
Organisation:	DTU
Author	Sune Frankild
Contact	sune.frankild@get2net.dk
Availability	Available
Located	http://www.it.dtu.dk/~asytools/download.html
Date	01/12/99
Maintenance	Maintained?
Comment	

Name:	NCL-VHDL
Organisation:	Theseus Logic
Author	Theseus Logic
Contact	support.eda@theseus.com
Availability	Available
Located	http://www.it.dtu.dk/~asytools/download.html
Date	01/12/99
Maintenance	Maintained?
Comment	Uses Synopsys Design Tools

Name:	PIPEFITTER
Organisation:	Politecnico di Torino & Universita di Udine
Author	Ivan Blunno
Contact	blunno@linus.polito.it
Availability	Available
Located	http://polimage.polito.it/~blunno/pipefitter/pipefitter.tar.gz
Date	28/05/98
Maintenance	Maintained
Comment	

Notation to Notation Conversion

Name:	TAST
Organisation:	TIMA-CMP
Author	Marc Renaudin
Contact	marc.renaudin@imag.fr
Availability	Not Available
Located	
Date	
Maintenance	Unknown
Comment	

Name:	DI2PN
Organisation:	SBU
Author	Dennis Furey
Contact	fureyd@sbu.ac.uk
Availability	Available
Located	http://www.sbu.ac.uk/~fureyd/di2pn/
Date	23/12/00
Maintenance	Maintained
Comment	

Name:	VL2ASTG
Organisation:	Politecnico di Torino & Universita di Udine
Author	Ivan Blunno
Contact	blunno@linus.polito.it
Availability	Available
Located	http://polimage.polito.it/~blunno/vl2astg/vl2astg.tar.gz
Date	05/06/00
Maintenance	No longer maintained
Comment	Subsumed by Pipefitter

XDI Tools

Name:	DIGG
Organisation:	RUG
Author	Willem Mallon
Contact	willem.mallon@philips.com
Availability	Unknown
Located	
Date	
Maintenance	Unknown
Comment	Tools removed from RUG's ftp server. Willem now at Philips.

Name:	LUDWIG
Organisation:	RUG
Author	Willem Mallon
Contact	willem.mallon@philips.com
Availability	Unknown
Located	
Date	
Maintenance	Unknown
Comment	Tools removed from RUG's ftp server. Willem now at Philips.

Name:	STEFFI
Organisation:	RUG
Author	Willem Mallon
Contact	willem.mallon@philips.com
Availability	Unknown
Located	
Date	
Maintenance	Unknown
Comment	Tools removed from RUG's ftp server. Willem now at Philips.

Verifiers

Name:	ANALYZE
Organisation:	University of Calgary
Author	Willem Mallon
Contact	willem.mallon@philips.com
Availability	Unknown
Located	
Date	
Maintenance	Unknown
Comment	Tools removed from RUG's ftp server. Willem now at Philips.

Name:	AVER
Organisation:	Stanford Unversity
Author	David Dill
Contact	dill@cs.stanford.edu
Availability	Unknown
Located	
Date	
Maintenance	Unknown
Comment	No information available. Dill currently working on Murphi verifier, may be applicable

Name:	BMC, CSML, CV, MCB, SMV
Organisation:	CMU
Author	Sergey Berezin
Contact	Sergey.Berezin@cs.cmu.edu
Availability	Unknown
Located	http://www.cs.cmu.edu/~modelcheck/[BMC
Date	BMC - 16/06/99 CSML/MCB - 29/07/98 SMV - 06/11/00
Maintenance	Maintained
Comment	

Name:	CONCURRENCY WORKBENCH
Organisation:	Edinburgh University
Author	Faron Moller
Contact	Perdita.Stevens@dcs.ed.ac.uk
Availability	Available
Located	http://www.dcs.ed.ac.uk/home/cwb/CWBEXPORTDIR/
Date	18/07/99
Maintenance	Maintained
Comment	CCS based verifier

Name:	CIRCAL SYSTEM
Organisation:	Uni.South Aust.
Author	Antonio Cerone
Contact	cowie@cs.unisa.edu.au
Availability	Available
Located	http://www.acrc.unisa.edu.au/~circal/
Date	
Maintenance	Maintained
Comment	Based on Circal process algebra

Name:	FIREMAPS
Organisation:	McGill University
Author	Radu Negulescu
Contact	radu@macs.ece.mcgill.ca
Availability	Online Demo only
Located	http://www.macs.ece.mcgill.ca/cgi-bin/cgiwrap/fm/demo_in.cgi
Date	
Maintenance	Maintained
Comment	

Name:	LARCH PROVER
Organisation:	MIT
Author	Steven Garland?
Contact	garland@lcs.mit.edu
Availability	Availaible
Located	ftp://ftp.sds.lcs.mit.edu/pub/Larch/lp/
Date	Dated:06/11/00
Maintenance	Unknown
Comment	Used by ST

Name:	LOTOS/CADP
Organisation:	INRIA/VASY
Author	CADP team
Contact	cadp@inrialpes.fr
Availability	Availaible - Licensed
Located	http://www.inrialpes.fr/vasy/cadp.html
Date	
Maintenance	Maintained
Comment	Used by Technion for verificationof Async Circuits.

Name:	RAINBOW
Organisation:	Manchester University
Author	Barringer,Gough, et al
Contact	alanw@cs.man.ac.uk
Availability	Available
Located	ftp://rainbowu@ftp.cs.man.ac.uk/version3.03/Solaris/rainbow303core-sunos5.tar.gz
Date	12/04/99
Maintenance	Maintained
Comment	http://www.cs.man.ac.uk/fmethods/projects/AHV-PROJECT/ahv-project.html

Name:	SPHINX
Organisation:	USC
Author	Vida Vakilotojar
Contact	vivakil@hala.usc.edu
Availability	Available
Located	http://jungfrau.usc.edu/SPHINX/SPHINX/
Date	15/09/00
Maintenance	Maintained
Comment	

Name:	VERACI
Organisation:	Cambridge University
Author	Paul Cunningham
Contact	pac22@cl.cam.ac.uk
Availability	Not Available
Located	
Date	
Maintenance	Maintained
Comment	Incomplete and unavailable, at present. Unsure of intentions

Name:	VERDECT
Organisation:	University of Waterloo
Author	Jo Ebergen
Contact	Jo.Ebergen@eng.sun.com
Availability	Not Available
Located	
Date	
Maintenance	Unknown
Comment	Unable to find any reference to it at Waterloo. Ebergen has moved to Sun

Name:	VERSIFY
Organisation:	UPC
Author	Oriol Roig
Contact	oriol.roig@theseus.com
Availability	Available
Located	ftp://ftp.ac.upc.es/pub/archives/cad/versify/
Date	06/11/00
Maintenance	Unknown
Comment	Oriol Roig has moved to Theseus. http://www.ac.upc.es/vlsi/versify/

Name:	VIS
Organisation:	Berkley
Author	Robert Brayton
Contact	brayton@eecs.berkley.edu
Availability	Available
Located	ftp://ic.eecs.berkeley.edu/pub/Vis/
Date	25/09/98
Maintenance	Unknown
Comment	Works in cooperation with SIS. http://www-cad.eecs.berkeley.edu/~vis

Handshake/Timing Validators

Name:	HORN
Organisation:	Amulet Group, Manchester University
Author	Rhodri Davies
Contact	
Availability	Available
Located	http://www.cs.man.ac.uk/amulet/projects/horn/
Date	23/05/96
Maintenance	Not maintained
Comment	Collection of tools, but Rhodri has left Amulet

Name:	TIMEVER
Organisation:	Olso University
Author	Per Arne Karlsen
Contact	Per Torstein Roine{perr@ifi.uio.no}
Availability	Available
Located	http://www.ifi.uio.no/~vlsi/async/timver/timver-1.0.tar.gz
Date	04/11/99 ?
Maintenance	Unknown
Comment	Per Karlsen has left. Suggest contacting his supervisor Per Torstein Roine

Simulators/Modelling Tools

Name:	FSIMAC
Organisation:	Isical, Intel
Author	Susmita Sur Kolay
Contact	ssk@isical.ac.in
Availability	Not Available
Located	
Date	
Maintenance	Unknown
Comment	Relatively recent. Fault Simulator

Name:	LARD
Organisation:	AMULET group, University of Manchester
Author	Phill Endecott
Contact	lard@cs.man.ac.uk
Availability	Available
Located	ftp://ftp.cs.man.ac.uk/pub/amulet/lard
Date	11/09/00
Maintenance	Maintained
Comment	

Name:	TESTIFY
Organisation:	UPC
Author	Oriol Roig
Contact	lard@cs.man.ac.uk
Availability	Available
Located	ftp://ftp.ac.upc.es/pub/archives/cad/versify/
Date	11/09/00
Maintenance	Maintained
Comment	Related to the versify suite

Interchange Languages

Name:	AND/IF
Organisation:	SUN, TUE
Author	Bob Sproull
Contact	rsproull@east.sun.com
Availability	
Located	
Date	
Maintenance	
Comment	

Descriptive Languages

Name:	Synchronized Transitions
Organisation:	DTU
Author	Jan Staunstrup
Contact	jsp@imm.dtu.dk
Availability	Not available
Located	
Date	
Maintenance	Unknown
Comment	Staunstrup has left DTU. Suggest contacting Jan Sparso.

Name:	HOP
Organisation:	University of Utah
Author	Ganesh Gopalakrishnan
Contact	ganesh@cs.utah.edu
Availability	
Located	
Date	
Maintenance	Unknown
Comment	Little information

Miscellaneous

Name:	AVEMAP
Organisation:	USC
Author	Wei-Chun Chou
Contact	wchou@eng.sun.com
Availability	Not Available
Located	http://jungfrau.usc.edu/wchou/avemap.html
Date	15/06/99
Maintenance	Unknown
Comment	Technology Mapper for BMMs and One hot Domino ccts. Chou has left USC.

Name:	USC-PET
Organisation:	USC
Author	Aiguo Xie
Contact	pabeerel@eiger.usc.edu
Availa ˇlity	Not Available
Located	Located:http://jungfrau.usc.edu/USC-PET/index.html
Date	
Maintenance	Unknown
Comment	Performance Estimator. No address for Xie, try Beerel.

Additional Contacts

Name:	Asynchronous Digital Design
Contact	rich@avlsi.com

Name:	University of Tokyo
Contact	nanya@hal.rcast.u-tokyo.ac.jp

Name:	NTT (Network Inovation Laboratories)
Contact	rysuke@exa.onlab.ntt.co.jp

Name:	Marly Roncken
Contact	mroncken@scdt.intel.com

Appendix B: Original Questionnaire

Hello, I am contacting you because you or an associated group has been identified as working in the area of asynchronous circuit techniques and/or systems. The European Commission, under the aegis of its ACiD-WG IST programme wishes to promote asynchronous technology and has commissioned a report “Design, Automation and Test for Asynchronous Circuits and Systems” which will describe the state-of-the-art in methods and tools for the design of asynchronous digital VLSI systems. The report is intended to be primarily of use to companies who are aware of the potential benefits of asynchronous circuit technology, but who need to know more about available asynchronous design methods and tools before committing resources.

It is intended to regularly update the information contained in this report and produce an annual public overview of the status of asynchronous design in industry.

If you are involved in the development of more than one tool (or are an originator of more than one methodology) please reply multiple times to this questionnaire.

The published form of the report will be in a form that is easily digestible by the target audience, ideally a single A4 page per tool/methodology. The report will be delivered in electronic form via the web. In order to assemble this report, and to evangelise your work to wider audience, I would be grateful if you would respond to the following questionnaire. Please reply by email to

acid-report@cs.man.ac.uk

- Doug Edwards

ASYNCHRONOUS TOOLS QUESTIONNAIRE

1) Please compose a summary of your tool or methodology including, but not limited to: (This summary should occupy not more than a single A4 page. Please provide easily readable file formats, plain test preferred, diagrams may be redrawn and should be submitted in common formats such as JPEG, PNG, EPS. Proprietary wordprocessor formats (e.g. Word, Framemaker) are unwelcome.)

- the name of the tools and methodologies
- strengths and weaknesses of the tool/methodology with respect to other asynchronous and synchronous tools and methodologies
- its application domain (high speed, low power embedded systems ...)
- the use of existing HDLs within your tool/methodology
- the extent of automation used

Separate to your summary we will include details from your answers to the questions given below:

2) Categorise the problem domain addressed by your tool/methodology from the list given below. Choose one category or sub-category favouring those at the top of the list (e.g. systems which can synthesise and simulate should be categorised as synthesisers):

- Synthesis
  - Timing driven/layout assisting
  - Burst-mode
  - STG/Petri net
  - Silicon compilers
  - Other
- Formal verifiers/Theorem provers
  - Using existing frameworks (CCS/CSP/CIRCAL/LOTOS...)
  - Other
- Timing validators (post-simulation/static timing analysis)
- Notation to notation conversion
  - Simulation front ends
- Simulators/Modelling tools

3) Describe the design flows used with your tool/by your organisation with your methodology. Highlight those commercial EDA tools used.

4) What is your approach to production testing of designs produced using your tool/ methodology (if appropriate)? Manually performed test engineering should also be described as well as any locally developed tools and commercially available test tools.

5) What is the current status of the tool and your asynchronous circuit design involvement? Please include:

- your current activities.
- the identity of the maintainer/developer of the tool or a contact point for more information
- tool availability including licensing, approximate cost (if applicable) and a contact address/URL for downloads or sales
- your future plans.

6) Detail any significant demonstrators of your approach. References to papers or products produced with their notable asynchronous advantages would be ideal.