μSystems Research Group
School of Electrical and Electronic Engineering
Merz Court
Newcastle University
Newcastle upon Tyne, NE1 7RU, UK

http://async.org.uk/

A survey of theory and practice in compositional design of asynchronous circuits

Jonny Beaumont

j.r.beaumont@ncl.ac.uk

NCL-EEE-MICRO-MEMO-2015-011

November 2015

Abstract

There are several methods of describing systems in a modular fashion. These descriptions may represent a system in one of several ways, for example in a text form or as a graph. By using a modular representation, a large system can be split into multiple smaller sections, which can make operations such as verification of a system simpler and quicker.

Multiple methods of designing asynchronous circuits also exist, and each one has benefits for designing a certain type of asynchronous circuits, and therefore has several pitfalls for designing another type of circuit.

Concepts [32] are a method of describing signal interactions in an asynchronous system, which can then be composed with other concepts, which produce Signal Transition Graphs. Multiple graphs can then be combined and used to produce a full specification of an asynchronous system. This method uses a modular representation of systems to help in the design of an asynchronous system.

This document discusses multiple methods of modular description and existing asynchronous design methods, including concepts, and compares these based on several metrics, in an attempt to see what the features of each method are, and how concepts fit in, and what further features could be added to concepts in the future.

Introduction

Tables 1 and 2 describe several modular methods for various purposes, including design and verification. Following this, Table 3 features a comparison of these methods, as well as *concepts*, the proposed method. This comparison is based on several metrics.

Tables 4 and 5 describe several existing asynchronous design methods. The tables following, Tables 6 and 7, compare these, as well as the proposed method, based on several metrics.

Finally, Tables 8 and 9 feature discussions of all of the modular methods and existing design methods featured in this document. The metrics used for comparison are as follows:

Asynchronous circuit support

Do the methods feature support for asynchronous circuits? This is only applied to modular methods, as all the design methods described are for asynchronous systems.

Software tool support

Which of these methods has some form of software which can assist in the design of a circuit? The proposed method at this stage has some limited tool support, but a tool is in development to automate the process.

Composition

For modular methods only, this metric shows whether a method allows for composition of smaller elements of a design.

Gate-level design

Can the existing design methods allow for logic gates to be designed? Can these then be referenced to abstract the complexity of the gates when designing systems?

Event-level design

With the design methods, is it possible to design a system based on events which occur in a system or the environment, such as signal transitions on inputs?

Protocol-level design

Can the listed design methods allow for signal protocols, such as handshakes, to be described and then be used in abstract to avoid repetition of the causal relationships between these signals?

Design focus

Asynchronous circuits can be little digital focused, for example a control system, which interacts with and aims to control analogue circuitry using a control signals and sensors. Asynchronous circuits can also be big digital focused, where they are aimed at data operations, with wires which are multiple bit widths. For only the design methods from Table 3, this metric will explain the main focuses of the design methods listed.

Title	Authors	University	Description	Ref
Algebra of parameterised graphs	Andrey Mokhov, Victor Khomenko	Newcastle University	Algebra of Parameterised Graphs [26] has been introduced to overcome limitations of CPOGs, such as the lack of structural abstraction and composition methods, as well as the difficulty of formal analysis and verification. Similarly to CPOGs, PGs target little digital systems and support signal and gate level modelling of asynchronous circuits. PGs have a moderate support in Workcraft [28, 1]	[26] [1]
Algebra of switching networks	Andrey Mokhov	Newcastle University	Algebra of Switching Networks [24] specifically addresses signal and transistor level design of little digital circuits. The key differentiating feature of this modelling approach is that both structure and behaviour of a system can be captured by the same mathematical expression and therefore both analysis and synthesis tasks can be achieved by rewriting the expression according to specific sets of rules. This modelling method supports various forms of composition, however there is currently no tool support.	[24]
Compositional models of distributed systems	Eric Fabre	Cité Universitaire Beaulieu, France	A method of describing distributed systems as a set of interacting elementary components. There is no global synchronisation, all components are self contained and run concurrently to one another, and interact by sharing variables. These are composed to provide the operation of the whole system. State estimation can be performed on each component and viewed, and because these components are small, this avoids the possibility of state explosion which can occur when performing this on the full system.	[9]
Compositional Verification	David E. Long	Carnegie Mellon School of Computer Science, Pennsylvania, USA	This verification approach aims at taking an asynchronous system design, and *decomposing* it into a set of components which all run in parallel. Each of these components is then verified for certain local properties separately, and the result of these will determine the integrity of the full specification. This also extends into *abstraction,* where a model is simplified before verification to produce an abstract model. This makes the verification process more efficient, providing that there is a relationship between the original component model and the abstract which proves that the verification of the abstract model will correctly verify the original model.	[17]
Conditional Partial Order Graphs	Andrey Mokhov	Newcastle University	Conditional Partial Order Graphs (CPOGs) [27, 23]target a class of systems that are comprised of multiple acyclic behavioural scenarios, such as microprocessors [25]. CPOGs are equipped with powerful scenario-level composition techniques that are automated in Workcraft [28, 21, 1]. Structural composition of CPOGs is very limited and not automated at present. The CPOG model has been extended to model asynchronous circuits with cyclic scenarios [22] at the levels of signals and gates, however, automation in this context is limited at present.	[23] [27] [25] [21] [22] [1]

Table 1: Description of modular, concept-like methods

Title	Authors	University	Description	Ref
DI Algebra	M.B. Josephs	Oxford University, UK	A method of describing systems as algebraic equations, specifying causal relations between signal transitions, making it ideal for asynchronous control systems. Each equation represents an operation of the specification, and these can be composed and simplified for a more compact version. All equations can then be composed to find an equation for the whole specification and again simplified for a more compact version.	[13]
Hierarchical design of DI systems	P.N. Lam H.F. Li	Department of Computer Science, Concordia University	Provides a set of building blocks, either *Delay Insensitive* (DI) or *hybrid* (Non-DI) blocks, which are not necessarily individual logic gates. These are composed by describing the interconnections between the blocks, and this forms a module. Modules can then be composed with other modules in the hierarchy. Signal Transition Graphs are used in this method for specification of a circuit from blocks and modules, and this can be used for analysis of the circuit.	[15]
Resynthesis	Arseniy Alekseyev, Ivan Poliakov, Victor Khomenko, Alex Yakovlev.	Newcastle University, UK	A process of decomposing a full model and recomposing it of selective components to reproduce a smaller model. This can be used to reduce the number of signals to connect two separate models for example. This process is regularly used for optimisation of BALSA circuits (see Table 2).	[2]
Snippets	Igor Benko, Jo Ebergen	University of Waterloo, Canada and Sun Microsystems, USA	Smaller *state graph* models which are used to compose full state graphs of larger systems. *Snippets* describe the operation of a part of a system in terms of input and output alphabets, and in which ways these snippets can fail. When composed with other snippets it can produce a working system state graph model.	[6]
Structural Design	Craig Armenti	Zuken USA	Features re-usability of modular components. A component design can be used multiple times across full device designs in conjunction with several other circuit modules. These modules can be changed in some way without affecting how they are used in a full device and how they interact with other modules. This method aims at promote reuse of circuit designs across different full systems, and reducing the need for redesign of correctly working systems for each new device.	[3]

Table 2: (cont.) Description of modular, concept-like methods

Title	Authors	University	Asynchronous support	Software tool support	Composition
Concepts (Proposed Method)	Jonathan Beaumont, Andrey Mokhov, Danil Sokolov, Alex Yakovlev	Newcastle University	Yes	Limited (Yes)	Yes
Algebra of parameterised graphs	Andrey Mokhov, Victor Khomenko	Newcastle University	Yes	Yes	Yes
Algebra of switching networks	Andrey Mokhov	Newcastle University	Yes	No	Yes
Compositional models of distributed systems	Eric Fabre	Cité Universitaire Beaulieu, France	Yes	No	Yes
Compositional Verification	David E. Long	Carnegie Mellon School of Computer Science, Pennsylvania, USA	Yes	No	Yes
Conditional Partial Order Graphs	Andrey Mokhov	Newcastle University	Yes	Yes	No
DI Algebra	M.B. Josephs	Oxford University, UK	Yes	No	Yes
Hierarchical design of DI systems	P.N. Lam H.F. Li	Department of Computer Science, Concordia University	Yes	No	Yes
Resynthesis	Arseniy Alekseyev, Ivan Poliakov, Victor Khomenko, Alex Yakovlev.	Newcastle University, UK	Yes	Yes	Yes
Snippets	Igor Benko, Jo Ebergen	University of Waterloo, Canada and Sun Microsystems, USA	No	No	Yes
Structural Design	Craig Armenti	Zuken USA	No	Yes	No

Table 3: Comparison of modular, concept-like methods

Title	Authors	University	Description	Ref.
Balsa	Doug Edwards, Andrew Bardsley	Department of Computer Science, University of Manchester, UK	A design approach which features a REGISTER TRANSFER LANGUAGE (RTL) like language, similar to VHDL or VERILOG, which aims to produce both data-driven and control circuits. This approach closely follows the process of the *Phillip's Tangram compiler*. A specification written in this language is used to produce a circuit implementation in two steps. First, a balsa program is converted into a format describing a network of handshaked components. This format can then be used for simulation, circuit diagrams and in the second step, which maps handshaked components on to library components for synthesis.	[8] [30]
Biscotti	Gang Jin, Lei Wang, Zhiying Wang	National University of Defense Technology Changsha, China	C-like language which features '*forever*' blocks, in which code runs sequentially, but all blocks run concurrently to each other. This design method starts by specifying a circuit. This is then *compiled* into formats for use by tools, Petri nets for verification in WORKCRAFT, for optimisation and net list generation. If these stages are successful, then the circuit can be synthesized. This is designed for data-driven asynchronous systems.	[12]
Caltech Synthesis Method	A.J. Martin	California Institute of Technology	Individual signal interactions, such as those for control systems, are specified using a regular expression style language, based on *Communicating Sequential Processes* (CSP). After these *programs* have been specified as a list of *processes*, they are then *compiled*, where a process is decomposed into a set of processes which are equivalent to the original. This occurs until all processes are in a simpler form that the compiler can continue to use. Next is *handshake expansion*, where handshaking replaces connections between each process. During this, some process orders may be changed which do not affect the operation, but may avoid issues such as deadlocks, this is known as *reshuffling*. Finally, *operator reduction* is performed to reduce the number of operators used in the new set of processes, by finding operators which can be described by other more standard operators. After this, the program will be synthesisable using a library of standard operations.	[19] [18] [11]
CHP	Alain J. Martin, Christopher D. Moore	Department of Computer Science California Institute of Technology, California, USA	*Communicating Hardware Processes* (CHP) is a programming language which is primarily used for designing asynchronous circuits. A program written in CHP consists of a fixed set of concurrent processes which communicate by messages. These processes are written separately, the code in each of which is usually sequential but some in-process concurrency is allowed, and a full system is produced from parallel composition of these processes. CHP processes use variables for data manipulation and for signal interactions, allowing standard programming constructs such as if..then statements for example which allow for selection of signals, useful for control systems. Processes do not share these variables however, and data is passed in messages through communication channels. There is a tool as part of CHP, called *CHPsim* which simulates CHP programs.	[20]

Table 4: Descriptions of existing design methods

Title	Authors	University	Description	Ref.
Cλash (Clash)	Christiaan Baaij	University of Twente, Netherlands	Another tool which uses HASKELL. There are similarities to LAVA; both feature built-in verification carried out in the same way, and users can define their own functions. However, CLASH has built in synthesis and simulation, avoiding the need to export VHDL for use in external tools, however this feature remains. A major difference is that in CLASH, some syntax of HASKELL is directly synthesisable as an asynchronous operation, for example a case statement can be used to specify choice in the circuit. To do this in LAVA, a user would have to compose functions in a certain way.	[14, 4]
Lava	Per Bjesse, Koen Claessen, Mary Sheeran	Chalmers University of Technology, Sweden	A tool written in the functional programming language HASKELL, with its own associated design flow able to design data-driven or control circuits, and all design steps can be performed in LAVA. It features several predefined functions, such as simple logic gates which can be used either as direct operations for circuits, or as part of user defined functions. A user can define a function in terms of inputs, operations on these inputs, and outputs. A circuit is defined as inputs, stored in variables, operations are performed on these using functions which can be sequential or parallel, and then variables are set as outputs. Lava has built in verification, using a parameter which defines verification property. This returns a logic equation which is automatically processed, returning a value determining whether the property is satisfied. Lava also features the ability to generate code for other languages, primarily VHDL, which can then be used by other tools for simulation and synthesis.	[7]
Proteus	Peter A. Beerel, Georgios D. Dimou, Andrew M. Lines	University of Southern California and Fulcrum Microsystems	Pipelines are the channels which pass data between stages of an asynchronous system which in some cases can cause *bottleneck*s, a major source of delay as data passage is slowed. PROTEUS is a tool which automatically analyses and optimises a pipelined system to reduce bottlenecks and delays. It takes in a RTL language or a CSP like specification. This is then synthesized, producing a net list which is then analysed and optimised to produce a new pipelined implementation which will have the best performance.	[5] [10]
Tiempo	Alex Yakovlev, Pascal Vivet, Marc Renaudin	Tiempo, France	TIEMPO introduced a design flow which uses a tool called ASYNCHRONOUS CIRCUIT COMPILER (ACC). This tool uses VERILOG which is used to model operations and communication channels between asynchronous entities, which are normally handshaked. The tool allows the use of several asynchronous architecture types aimed at data-driven circuits, i.e. pipelined, parallel, sequential etc. Synthesis uses libraries which contain asynchronous cells, and constraints such as timing information have to be specified for us by the tool . First, a net list is produced and further constraints produced by the tool. A *place and route* tool then optimises and verifies the system based on the constraints, and a few necessary properties.	[31]
Uncle	Robert B. Reese, Scott C. Smith, Mitchell A. Thornton	Mississippi State University, University of Arkansas, Southern Methodist University	A tool which uses a design approach aimed at producing an implementation using *Null Convention Logic* (NCL), a set of components which have a state similar to *precharge.* Each component starts in the null state where outputs and inputs are all null, which does not represent any data. It remains in this state until data is present on all inputs, at which point the component will output data based on the inputs. This data will be held on the output of the component until all inputs return to null, when the output will return to null. UNCLE uses an RTL language. This tool then synthesises the specification using a library of NCL components which can be simulated and verified, ultimately producing an NCL implementation.	[29] [16]

Table 5: (cont.) Descriptions of existing design methods

Title	Authors	University	Tool support	Gate-level design	Event-level design	Protocol-level Design	Design focus
Concepts (Proposed Method)	Jonathan Beaumont, Andrey Mokhov, Danil Sokolov, Alex Yakovlev	Newcastle University	Limited (Yes)	Yes	Yes	Yes	Little digital
Algebra of parameterised graphs	Andrey Mokhov, Victor Khomenko	Newcastle University	Yes	Yes	Yes	No	Little digital
Algebra of switching networks	Andrey Mokhov	Newcastle University	No	Yes	No	No	Little digital
Balsa	Doug Edwards, Andrew Bardsley	Department of Computer Science, University of Manchester, UK	Yes	Yes	No	Yes	Big digital
Biscotti	Gang Jin, Lei Wang, Zhiying Wang	National University of Defense Technology Changsha, China	Yes	Yes	No	No	Big digital
Caltech Synthesis Method	A.J. Martin	California Institute of Technology	No	Yes	Yes	Yes	Little digital
CHP	Alain J. Martin, Christopher D. Moore	Department of Computer Science California Institute of Technology, California, USA	Yes	No	Yes	Yes	Big digital

Table 6: Comparison of existing design methods

Title	Authors	University	Tool support	Gate-level design	Event-level design	Protocol-level Design	Design focus
Cλash (Clash)	Christiaan Baaij	University of Twente, Netherlands	Yes	Yes	No	Yes	Big digital
Conditional Partial Order Graphs	Andrey Mokhov	Newcastle University	Yes	Yes	Yes	No	Little digital
Lava	Per Bjesse, Koen Claessen,Mary Sheeran	Chalmers University of Technology, Sweden	Yes	Yes	No	Yes	Big digital
Proteus	Peter A. Beerel,Georgios D. Dimou, Andrew M. Lines	University of Southern California and Fulcrum Microsystems	Yes	Yes	No	No	Big digital
Tiempo	Alex Yakovlev, Pascal Vivet, Marc Renaudin	Tiempo, France	Yes	Yes	No	Yes	Big digital
Uncle	Robert B. Reese, Scott C. Smith,Mitchell A. Thornton	Mississippi State University, University of Arkansas, Southern Methodist University	No	Yes	No	No	Big digital

Table 7: (cont.) Comparison of existing design methods

Title	Authors	Comparison	Ref.
Balsa	Doug Edwards, Andrew Bardsley	RTL languages are regularly used for synchronous design, thus a designer can adapt more easily to asynchronous design. These languages feature the ability to easily perform operations on multiple bits unlike the proposed approach, and uses programming constructs such as conditional statements for control. Specifying a control system can lead to a complicated program which can be difficult to comprehend, in comparison to the STGs produced in the proposed approach, in which signal interactions can be visualised. RTL languages do allow for reuse of modules, something we aim to address with the proposed method, and this can speed up the design process.	[8] [30]
Biscotti	Gang Jin, Lei Wang, Zhiying Wang	Similar to BALSA, a C-like language can be easy to adapt to, as designers are likely to have programming knowledge. BISCOTTI, however is designed primarily for data-driven circuits, for specifying data operations which run in parallel, and communications between them. As with BALSA and RTL languages in general, specifying an asynchronous control system can become complex, the more signals there is to describe, however, reuse of written code can help to produce a quicker design process.	[12]
Caltech Synthesis Method	A.J. Martin	As with the proposed method, the Caltech Synthesis Method is used to describe causalities at the level of signal transitions. Because of the CSP language, understanding a program can be complicated if there are more than a handful of signals, and while writing a specification is somewhat simpler than in that of an RTL language, reusing a CSP specification is not as simple, nor as simple as with the reuse of concepts and scenarios in the proposed method.	[19] [18] [11]
CHP	Alain J. Martin, Christopher D. Moore	CHP provides an easier language for specifying asynchronous circuits than those methods which use an RTL language, and CHP is popular for this reason. Specifying control through signal states is simpler, and data operations can be specified also. It is similar to BISCOTTI in that blocks of sequential code are written, and these all run concurrently, but CHP offers simpler methods of specifying the communication channels between these blocks. Reuse is therefore available in CHP, but a control system featuring many signals and interactions can still be difficult to specify, comprehend and debug.	[20]
Cλash (Clash)	Christiaan Baaij	HASKELL is a functional language, and has a greatly differing syntax to that of C, or any RTL language. This can therefore be hard to learn. CLASH has some benefits over LAVA and similarities to VHDL and VERILOG such as syntax directly mapping to asynchronous operations. Specified components can be reused which can be useful, but HASKELL is not widely used with various existing design tools, and while CLASH does feature built in tools for verification, simulation and synthesis, it can be hard to integrate this with other existing methods.	[14, 4]
Compositional models of distributed systems	Eric Fabre	This method, while not used to design asynchronous circuits, features similar ideas to scenarios from the proposed method. Splitting a system and verifying them separately can be useful for both efficiency, and for debugging.	[9]
Compositional Verification	David E. Long	As with Compositional Models of Distributed systems, this methodology aims at decomposing a full system into several smaller components, and verifying these separately. Our proposed method is similar to this, but rather than decomposing a full system, we aim to design from the ground up into several smaller scenarios, verifying these and then combining them to produce a full system specification.	[17]
DI Algebra	M.B. Josephs	The proposed method is similar to DI algebra, however concepts are described textually, which is different to DI algebra and as such, simplification does not occur at concept level, but during the composition and combination steps. To the best of our knowledge there are no tools or methodologies supporting compositional design of asynchronous circuits based on DI algebra and thus it is incompatible with the rest of our design flow and not suitable for use in industrial settings.	[13]

Table 8: A comparison of the proposed method, and all methods mentioned above

Title	Authors	Comparison	Ref.
Hierarchical design of DI systems	P.N. Lam H.F. Li	Hierarchical design of DI systems is very similar to the proposed method. Both feature a lower level of specification (building blocks or concepts) which is used to create an STG specification (modules or scenarios) and these are then combined in some manner to produce a full system specification. The difference is that the building blocks provided in the hierarchical design method are set, and our proposed method allows for users to define their own low level specification components.	[15]
Lava	Per Bjesse, Koen Claessen, Mary Sheeran	Due to LAVA being a HASKELL based language, it features similar issues to CLASH. It features fairly different syntax to languages used in other existing methods. Unlike CLASH, LAVA does not feature built in tools for synthesis and simulation, and as such, specifications must be converted into VHDL for these processes, a feature which is built in.	[7]
Proteus	Peter A. Beerel, Georgios D. Dimou, Andrew M. Lines	PROTEUS takes in a specification in the form of CSP or VHDL, which can be useful for designers who may prefer one language over another, however PROTEUS is a tool which analyses and optimises a pipelined system which are generally data-driven systems, and as such is used for the design and optimisation of a different type of asynchronous circuit to the proposed method.	[5] [10]
Resynthesis	Arseniy Alekseyev, Ivan Poliakov, Victor Khomenko, Alex Yakovlev.	This process is regularly used for optimisation of BALSA control circuits, however in BALSA the set of predefined components is fixed, so a designer cannot easily introduce new scenarios. Resynthesis requires full models which are decomposed. For the proposed methodology we take a ground-up approach to design, starting with concepts to be composed, producing scenarios which are combined into a complete model. Resynthesis can be used at a later stage of the proposed approach, once the complete model of a system (or a subsystem) has been obtained.	[**?**]
Snippets	Igor Benko, Jo Ebergen	Snippets specify the operation of part of a system in terms of input and output alphabets, showing the outputs produced from the inputs based on the state, where as with our proposed approach we want to go deeper than this and compose a component from concepts which are responsible for capturing signal behaviours for system features, such as handshakes, mutual exclusion, synchronisation, etc.	[6]
Structural Design	Craig Armenti	The ideas of this method are similar to that of the design method we are proposing to reduce design time by reusing previously designed elements. However, this method is at a much higher level, using fully designed and tested components where as we propose to allow reusability of when modelling at circuit level, using composed concepts.	[3]
Tiempo	Alex Yakovlev, Pascal Vivet, Marc Renaudin	TIEMPO uses VERILOG as a specification language, another RTL language. This has some differences to design flows like BALSA, mainly in how it verifies and synthesises a specification, but the advantages and disadvantages are similar. These design methods are usually used for data-driven systems, making them unsuitable for control systems.	[31]
Uncle	Robert B. Reese, Scott C. Smith, Mitchell A. Thornton	UNCLE also uses an RTL language for specification of circuits, and these are discussed above. In UNCLE, specifying a circuit is carried out in effectively the same way as other methods, the differences are in the synthesis and verification, where null convention logic is used which operates differently and requires different verification properties to standard logic types produced by other design tools.	[29] [16]

Table 9: (cont.) A comparison of the proposed method, and all methods mentioned above

References

[1] Workcraft framework webpage: www.workcraft.org.

[2] Arseniy Alekseyev, Ivan Poliakov, Victor Khomenko, and Alex Yakovlev. Optimisation of balsa control path using stg resynthesis. In *UK Asynchronous Forum*, 2009.

[3] Craig Armenti. Get to market faster with modular circuit design. *Electronic Engineering Journal*, 2015.

[4] Christiaan Baaij. Clash: From haskell to hardware. Master's thesis, University of Twente, 2009.

[5] P.A. Beerel, G.D. Dimou, and A.M. Lines. Proteus: An asic flow for ghz asynchronous designs. *Design Test of Computers, IEEE*, 28(5):36–51, Sept 2011.

[6] Igor Benko and Jo Ebergen. Composing snippets. In Jordi Cortadella, Alex Yakovlev, and Grzegorz Rozenberg, editors, *Concurrency and Hardware Design*, volume 2549 of *Lecture Notes in Computer Science*, pages 1–33. Springer Berlin Heidelberg, 2002.

[7] Per Bjesse, Koen Claessen, Mary Sheeran, and Satnam Singh. Lava: hardware design in haskell. In *ACM SIGPLAN Notices*, volume 34, pages 174–184. ACM, 1998.

[8] D. Edwards and A. Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[9] Eric Fabre. Compositional models of distributed and asynchronous dynamical systems. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 1, pages 1–6. IEEE, 2002.

[10] G. Gill and M. Singh. Automated microarchitectural exploration for achieving throughput targets in pipelined asynchronous systems. In *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, pages 117–127, May 2010.

[11] Charles Antony Richard Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[12] Gang Jin, Lei Wang, and Zhiying Wang. A new description language for data-driven asynchronous circuits and its design flow. In *Circuits, Communications and Systems, 2009. PACCS '09. Pacific-Asia Conference on*, pages 322–325, May 2009.

[13] M.B. Josephs and J.T. Udding. An overview of d-i algebra. In *System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on*, volume i, pages 329–338 vol.1, Jan 1993.

[14] Jan Kuper and Christiaan Baaij. Hardware specification with cλash. *DSL 2013*, 2013.

[15] PN Lam and HF Li. Hierarchical design of delay-insensitive systems. *Computers and Digital Techniques, IEE Proceedings E*, 137(1):41–56, 1990.

[16] Michiel Ligthart, Karl Fant, Ross Smith, Alexander Taubin, and Alex Kondratyev. Asynchronous design using commercial hdl synthesis tools. In *Advanced Research in Asynchronous Circuits and Systems, 2000.(ASYNC 2000) Proceedings. Sixth International Symposium on*, pages 114–125. IEEE, 2000.

[17] David E Long. *Model checking, abstraction, and compositional verification*. PhD thesis, Citeseer, 1993.

[18] A. Martin. Compiling communicating processes into delay-insensitive vlsi circuits. *Distributed Computing, vol. 1(4)*, pages 226–234, 1986.

[19] Alain J Martin. A synthesis method for self-timed vlsi circuits. *IEEE Computer*, pages 224–229, 1987.

[20] Alain J Martin and Christopher D Moore. Chp and chpsim: A language and simulator for fine-grain distributed computation. Technical report, Caltech Technical Report CS-TR-1-2011, 2011.

[21] A Mokhov, A Alekseyev, and A Yakovlev. Encoding of processor instruction sets with explicit concurrency control. *Computers & Digital Techniques, IET*, 5(6):427–439, 2011.

[22] A. Mokhov, D. Sokolov, and A. Yakovlev. Adapting asynchronous circuits to operating conditions by logic parametrisation. *2012 IEEE 18th International Symposium on Asynchronous Circuits and Systems*, pages 17–24, 2012.

[23] Andrey Mokhov. *Conditional Partial Order Graphs*. PhD thesis, Newcastle University, 2009.

[24] Andrey Mokhov. Algebra of switching networks. *IET Computers & Digital Techniques*, 2015.

[25] Andrey Mokhov, Alexei Iliasov, Danil Sokolov, Maxim Rykunov, Alex Yakovlev, and Alexander Romanovsky. Synthesis of processor instruction sets from high-level isa specifications. *IEEE Transactions on Computers*, 63(6):1552–1566, 2014.

[26] Andrey Mokhov and Victor Khomenko. Algebra of parameterised graphs. *ACM Transactions on Embedded Computing Systems*, 13(4s), 2014.

[27] Andrey Mokhov and Alex Yakovlev. Conditional partial order graphs: Model, synthesis, and application. *IEEE Transactions on Computers*, 59(11):1480–1493, 2010.

[28] Ivan Poliakov, Victor Khomenko, and Alex Yakovlev. Workcraft–a framework for interpreted graph models. In *Applications and Theory of Petri Nets (ATPN)*, pages 333–342. Springer, 2009.

[29] R. B. Reese, S. C. Smith, M. Thornton, et al. Uncle-an rtl approach to asynchronous design. In *Asynchronous Circuits and Systems (ASYNC), 2012 18th IEEE International Symposium on*, pages 65–72. IEEE, 2012.

[30] Kees Van Berkel. *Handshake circuits: an asynchronous architecture for VLSI programming*, volume 5. Cambridge University Press, 1993.

[31] Alex Yakovlev, Pascal Vivet, and Marc Renaudin. Advances in asynchronous logic: From principles to gals amp; noc, recent industry applications, and commercial cad tools. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1715–1724, March 2013.

[32] J. Beaumont A. Mokhov D. Sokolov A. Yakovlev. Compositional design of asynchronous circuits from behavioural concepts. In *ACM-IEEE International Conference on Formal Methods and Models for System Design MEMOCODE15*, June 2015.