High-Level Synthesis of GALS Systems

Mahdi Jelodari Mamaghani, Jim D. Garside
School of Computer Sceince
The University of Manchester
Manchester, UK M139PL
Email: mamagham@cs.man.ac.uk

***Abstract*—The aim of this research is to automate the synthesis process of synchronous elastic (SE) systems whilst exploiting the advantages of data-flow concurrency of asynchronous design. This approach automates the integration of synchrony and asynchrony. Therefore, it makes it possible to investigate high level synthesis of Globally Asynchronous Locally Synchronous (GALS) systems without the need to build trivial links and ports and the ad-hoc insertion of synchronisers etc. Our proposed method enables the designer to use a unified language to develop flexible multi-clocked SoCs.**

I. Introduction

The forward-looking design trend in Very Large Scale Integrated (VLSI) is Systems-on-Chip (SoC). SoC aims to integrate multiple computation, communication and storage components into a single chip and targets high performance systems by elimination of most off-chip communication costs. It is agreed that running SoC components under control of a single clock is not feasible and clock distribution has been revealed as a critical obstacle. For instance, It has been shown that the die-to-die clock frequency varies by 30% in 180nm technology [1]. This *variability* is responsible for systematic uncertainties in computation and communication delays. Therefore designers are forced to deal with variability by considering conservative margins [2] that could be prohibitive for the system to achieve its potential performance.

Robustness towards variation has been widely cited as an advantage of asynchronous circuits [3][4]. Although these circuits can benefit from fine grained elasticity, their complex handshake protocol imposes expensive implementation costs in terms of area. For instance, an asynchronous delay-insensitive (aka asynchronous elastic) circuit occupies almost four times the area of a conventional inelastic synchronous circuit [4]. Additionally, the lack of mature EDA tools in this domain has been an important reason why designers have been reluctant to adopt this paradigm.

A less radical solution is Globally Asynchronous Locally Synchronous (GALS) [5] which offers potential advantages in this respect, as it preserves system modularity and concentrates on communication aspects. The main problems of GALS design are the lack of methodology and tool support for partitioning the system into the synchronous islands. The existing GALS methods are of ad hoc nature, rather than being a process of synthesis with optimisation. Only rudimentary design automation has been proposed [6] where the top-level hierarchy determines the boundaries for synchronicity "islands" and a static analysis of communication between these islands defines the choice of fairly limited communication mechanisms. We propose an automated method to implement GALS systems at a higher abstraction level which is independent of technology, protocol, data encoding or any other details of circuit design.

Our approach takes advantage of synchronous elasticity [7] (aka SELF protocol) to introduce a common timing discipline to the circuit which transforms it into a latency insensitive system. A latency insensitive system is able to tolerate dynamic changes in the computation and communication delays. This feature enables us to raise the level of abstraction to the data-flow representation where functionality is separated from timing details. Therefore, it is possible for a designer to specify a large scale system by only concentrating on its functionality and postpone timing complexity to when synthesis takes place. Currently, research work is focused on the development of a synthesis flow to exploit GALSification techniques for partitioning the system into individual synchronous clusters.

Unlike many previous systems, our design flow employs a data-driven synthesis style to distribute controllers through the circuit which contributes to its modularity and enhanced concurrency. This facilitates partitioning into elastic blocks and is supposed to pave the road for further optimisations using commercial EDA tools.

II. Obstacles with traditional GALS design

Three decades ago Chapiro introduced the GALS concept to the community [5]. Since then academia and industry have attempted to exploit this concept in SoC design which can potentially benefit from the multi-clocked behaviour of the GALS design. Broadly speaking, all the efforts to exploit the GALS methodology fall in the lines of ad hoc design where synchronous blocks are glued together using trivial interfacing logic and synchronisers [8]. The tremendous challenge with the multi-clock design is the implementation of stable communication between clock islands. To combat this issue many techniques are proposed in the literature including pausible clock, asynchronous and loosely synchronous interfacing etc. These techniques address the metastability issues through explicit FIFO insertion which needs accurate considerations of timing details at circuit level and is survivable only by using the assemble-and-verify technique.

We believe that the time has arrived to automate the building of GALS systems without reflecting the extreme timing behaviour of the circuit to the designer. It is agreed that the synchronous approach with the corresponding accuracy complicates the design when it comes to SoC while the asynchronous approach simplifies the design aspects by separating timing from functionality [9]. Accordingly, we employ the asynchronous design approach along with an advanced

communication protocols to investigate the possibilities towards developing a new synthesis paradigm for GALS design. Section III explains our approach towards this objective.

III. Our approach

Our approach exploits the fine-grained data-flow concurrency inherent from the asynchronous design rather than just preserving the latency insensitivity. We aim at raising the design abstraction level from RTL to algorithmic level to provide the designer with a flexible implementation of concurrent hardware. At this level system functionality is specified by data flows, apart from timing constraints. In general, raising the level of abstraction could have three major benefits:

1) The designer is able to specify the hardware in the form of concurrent data flows rather than thinking in a sequential manner and squeezing the tasks in time boundaries.

2) It provides traditional designers with an interface to cover their unfamiliarity with asynchronous techniques, protocols or data-encoding in circuit implementation.

3) A higher level abstraction allows flexible exploration of the design space based on formal models, such as Petri nets [10] where it is possible to consider different analyses and measurements.

Regarding these advantages, we introduce eTeak as a high level synthesis framework for designing synchronous elastic systems [11]. eTeak is developed based on the Teak system [12] which is a token flow implementation for Balsa language [13]. It is capable of generating syntax-directed, data-driven handshake circuits for Balsa descriptions using a new component set to target delay-insensitive communication. In section IV we discuss the features in detail and explain them in the *GALSification* context.

IV. Teak: a data-flow synthesis system

We group the features of Teak into communication and computation facets. From the communication perspective the Teak networks are synthesised in a syntax-directed compilation manner from a CSP-like language. The primitives of the language, including channels and processes, are preserved which forms a **point-to-point communication** between the computation blocks at hardware level which contributes to concurrency and synchronous massage passing.

The networks are **slack elastic** [14] which means the communication channels are capable of storing any degree of tokens. This feature enables us to modify the level of pipelining over the channels without affecting the behaviour of the circuit.

From the computational perspective the network is built based on the **macro-module** style [15] with separate *go* and *done* activation signals. These modules are chained in sequence or parallel according to the source level directives. The macro-module architecture contributes to a distributed control mechanism where the datapath and the corresponding control are enclosed within a macro-module.

Accordingly, modules are controlled locally through handshaking so whenever data becomes available computation can start. This concept has already been introduced in **data-flow** systems [16]. Based on this concept data-dependent computation becomes possible which means that independent data streaming could exist within a module which can significantly influence the performance of the circuit. In addition, it allows the tool to perform functional decomposition over a module and define new boundaries.

V. Teak towards *GALSification*

We explain how the features of Teak are exploitable towards automating the GALSification process and multi-clocked SoC design.

1) Point-to-Point Communication: Point-to-Point (PTP) communication enables a module to have independent rates of data streaming from different sources which contributes to a higher level of concurrency and accordingly effective throughput. Let's assume that module A with the input set of {a,b,c} and the output set of {x,y} is capable of performing two functions f,g which are not necessarily independent. The function f takes {a,b} as input and g takes {b,c}. if we assume that input values are supplied with different rates of a', b' and c' where a' is the slowest rate then g can operate and produce output independent from a' which results in higher throughout of module A. The PTP communication is closely compatible with the data-flow style of computation of the modules which will be discussed later.

2) Slack Elasticity: A *Slack Elastic* system can be pipelined with any degree of storage on its communication channels. This behaviour was first formalised for the distributed computation systems which were described in a CSP-like language, CHP [14]. Slack Elasticity provides a flexible communication environment for the computational blocks in the system. We take advantage of this feature in Teak to optimise the processes without affecting the overall functionality of the system. Composition and decomposition of modules towards *GALSifcation* benefits from elastic communication which is not available in the synchronous domain where rigid timing controls the communications. The elastic behaviour is preserved when the SELF protocol is employed. In section VII we demonstrate a simple synchronous elastic processor which follows this concept.

3) Macromodule Style: The *Macromodule logic* was introduced to enable designers implement complex circuits using simple data processing building blocks [15]. Later, this concept was used to simplify the asynchronous control design [17]. Teak employs this technique to perform the control interactions locally instead of having them as a separate central unit which has significant performance implications. We exploit the local (aka distributed) control behaviour to perform functional composition and decomposition of Macromodules which results in defining new boundaries within the network. Moreover, the Macromodules allow us to optimise one module without affecting the behaviour of the network.

4) Dataflow Architecture: Dataflow machines emerged as an alternative design style to reduce the centralised control effect and speed up the computation by prioritising the data [16]. In the Teak networks dataflow architecture joined with PTP communication realises concurrency and eases the modules decomposition process. Decomposing the modules

towards *GALSification* based on their functionality, rather than structural properties, is our main objective.

Teak extracts parallel entities from the high-level Balsa code, produces a control-data flow graph (CDFG) and then maps it onto the Macromodules with local control handshaking through the go-done channels. Therefore the resulting circuit benefits from distributed control scheme. This feature allows us to explore different architectures by replacing the communication-heavy asynchronous designs with Finite State Machines (FSMs) which would trade off the elasticity and the concurrency level inherent in the asynchronous design.

VI. Synchronous Elastic Teak

In the light of the mentioned features, Teak was considered as a desired framework to inspect the *GALSification* process from a high level perspective. To incorporate synchronous elasticity in Teak, the existing component library is adapted to the SELF protocol and buffers are converted to time decoupling controllers to govern the flow of control and data based on the elastic protocol [11]. Since the new component set does not infer any combinational feedback loops within the network, conventional EDA can be used for optimisation purposes. Moreover, SELF is beneficial for the computation blocks as it simplifies the deadlock freedom issue with loops according to its simple interlocking behaviour.

VII. Results

As a case study Manchester Small-Scale Experimental Machine (SSEM) [18] is exercised. SSEM is a simple iterative processor. The Balsa description of it is synthesised using Teak and eTeak to generate the asynchronous and the synchronous elastic version of the machine respectively with the same level of granularity. In Figure 1 the area cost associated with truly asynchronous and synchronous elastic design styles is depicted. This experiment confirms that SELF preserves slack elasticity which is the key property for further investigation. Additionally, SELF simplifies the loop structures in the dataflow network and allows us to use synchronous CAD tools to optimise the circuit, particularly computation-heavy data manipulation units and detect the combinational loops for sake of deadlock freedom.

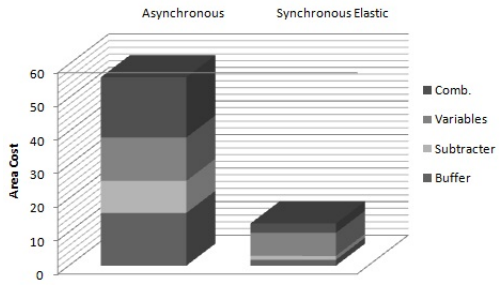

Fig. 1. The asynchronous dual-rail SSEM vs. its synchronous elastic counterpart in terms of cell area (UMC 130nm technology) which operates @ 740MHz. Each column is fragmented based on the existing entities in the circuit. Buffers are used to remove deadlock.

VIII. Conclusion and Future work

Our work successfully provides a flexible communication medium based on the synchronous elastic protocol for the computation in the circuit and enables us to exploit CAD tools for further transformations in the synchronous domain. Based on our analysis, elasticity at component level suffers from prohibitive costs in terms of performance as communication overhead dominates computation. A reasonable alternative is to replace them with Synchronous Sequential Machines (SSMs) and consequently, reduce the level of elasticity. Accordingly, our future work is to extend eTeak to be capable of decomposing modules into macromodules based on specific functionalities and consider optimising them using CAD tools. Thereafter we will focus on composing/grouping the modules based on their timing behaviour to form clocked islands.

IX. Acknowledgement

This work is supported by EPSRC Grant "Globally Asynchronous Elastic Logic Synthesis (GAELS)" (EP/I038306/1). The authors would like to thank Will Toms and Danil Sokolov for their help in refining the ideas presented in this paper.

References

[1] S. Borkar *et al.*, "Parameter variations and impact on circuits and microarchitecture," in *Design Automation Conference, 2003. Proceedings*, 2003, pp. 338–342.

[2] S. Hanson *et al.*, "Energy optimality and variability in subthreshold design," in *Low Power Electronics and Design, 2006. ISLPED'06. Proceedings of the 2006 International Symposium on*, 2006, pp. 363–365.

[3] L. T. Duarte, "Performance-oriented Syntax-directed Synthesis Of Asynchronous Circuits," 2010.

[4] J. Sparsø *et al.*, *Principles of asynchronous circuit design: a systems perspective*. Springer-Netherlands, 2001.

[5] D. M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Stanford University, 1984.

[6] A. Hemani *et al.*, "Lowering power consumption in clock by using globally asynchronous locally synchronous design style," in *Design Automation Conference, 1999. Proceedings. 36th*, 1999.

[7] J. Carmona *et al.*, "Elastic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[8] P. Teehan *et al.*, "A survey and taxonomy of gals design styles," *Design Test of Computers, IEEE*, 2007.

[9] K. S. Stevens *et al.*, "The future of formal methods and gals design," *Electronic Notes in Theoretical Computer Science*, vol. 245, 2009.

[10] C. A. Petri, "Kommunikation mit automaten," Ph.D. dissertation, Universitt Hamburg, 1962.

[11] M. J. Mamaghani *et al.*, "eteak: A data-driven synchronous elastic synthesiser," in *13th International Conference on Application of Concurrency to System Design, PhD Forum*, 2013.

[12] A. Bardsley *et al.*, "Teak: A token-flow implementation for the balsa language," in *Application of Concurrency to System Design, 2009. ACSD '09. Ninth International Conference on*, 2009.

[13] D. A. Edwards *et al.*, "Balsa: An asynchronous hardware synthesis language," *The Computer Journal*, vol. 45, 2002.

[14] R. Manohar *et al.*, "Slack elasticity in concurrent computing," in *Proceedings of the Fourth International Conference on the Mathematics of Program Construction*. Springer-Verlag, 1998.

[15] M. J. Stucki *et al.*, "Logical design of macromodules," in *Proceedings of the Joint Computer Conference*. ACM, 1967.

[16] Arvind *et al.*, "Annual review of computer science." Annual Reviews Inc., 1986, ch. Dataflow Architectures.

[17] J. Cortadella *et al.*, *Logic Synthesis for Asynchronous Controllers and Interfaces*. Springer, 2002.

[18] S. Lavington, "A History of Manchester Computers (2nd ed.), Swindon: The British Computer Society," 1998.