Exploiting Synchrony for Area and Performance Improvement in the Asynchronous Domain

Mahdi Jelodari Mamaghani, Will B. Toms, Andrew Bardsley, Jim D. Garside
School of Computer Science, The University of Manchester, UK M13 9PL
Email: mamagham@cs.man.ac.uk

***Abstract*—This work proposes a synthesis process called 'eTeak' which exploits synchronous EDAs to improve the implemented circuits. In this regard, it incorporates the synchronous elastic protocol in the Teak synthesis flow to move fine-grained concurrency from the asynchronous into the synchronous domain where clocked CAD tools can optimise the data manipulation units. A transformation technique is also proposed to enable the designer to explore the level of elasticity in the network and trade off the costs associated with computation and communication.**

I. Introduction

Asynchronous systems are well-known for their capability in clock-less communication and computation. In contrast with synchronous systems, asynchronous components rely on handshake signals for their interactions. Clock-less behaviour of asynchronous circuits contributes to several interesting properties, including channel-based communication which realises fine-grain concurrency, better modularity and composability, lower power consumption according to the capability of on-demand computation and average-case performance due to the possibility of using variable latency units. However, a lack of mature design tools has always been an obstacle to exploring the asynchronous domain design space.

In the past decade, the asynchronous community has proposed High Level Synthesis (HLS) tools, targeting large scale asynchronous circuit synthesis whilst hiding the signal level details of asynchronous design techniques, protocols or data-encoding. Among them, Balsa [1], Haste/TiDE and CAST/CHP [2] emerged as mature, full compilation systems which provide CSP-like languages to let designers specify circuits at higher abstraction level. Later, the community has attempted to address the performance issues, either by improving the control flow [3], [4] or extending the synthesis process [5].

In 2009, Teak [6] was proposed as a *dataflow* synthesis system to avoid the control overhead of Balsa[1]. A limited set of components is used to generate Teak Dataflow Networks (TDNs) in a syntax-directed fashion. The generated structures reflect the macro-module design style which simplifies the control tree. The dual-rail handshake protocol, employed for inter-component communication, imposes huge area overhead to the circuit. A year later, Peeters et al. announced a similar dataflow synthesis mechanism [7] which uses self-clocked templates to control the flow of data in the pipeline stage. Although these templates are free of latches and C-elements, specific timing assumptions are required to drive the flip-flops properly.

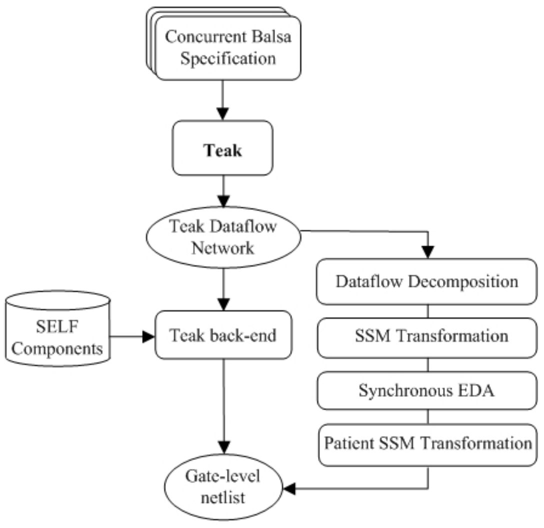

Fig. 1. eTeak synthesis flow

Our approach incorporates the Synchronous Elastic protocol (SELF) [8] in the Teak synthesis flow to reduce communication costs and enable the tool to exploit synchronous EDAs for logic synthesis whilst preserving the fine-grained concurrency and the network properties (section II). In addition, a transformation mechanism was derived to convert TDNs into the Synchronous Sequential State Machines (SSMs) which reduces the level of elasticity by partially removing the inter-component communications. In this context, a heuristic is being developed to partition the network based on its functional behaviour [9]. Figure 1 depicts the eTeak flow.

II. Teak: a data-flow synthesis system

Teak features divide into communication and computation aspects. From the communication perspective Teak networks are synthesised in a syntax-directed compilation manner from the Balsa language. The primitives of the language, including channels and processes, are preserved which form **point-to-point communication** between computation blocks at hardware level; this contributes to concurrency and synchronous massage passing.

The networks are **slack elastic** [10] which means the communication channels are capable of storing any number of tokens. This feature enables modification in 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 [11] with separate *go* and

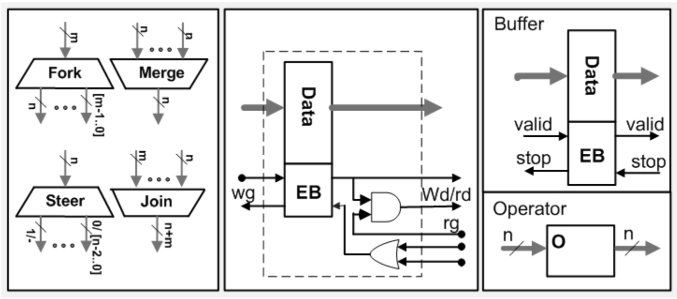

Fig. 2. eTeak SELF adapted component set

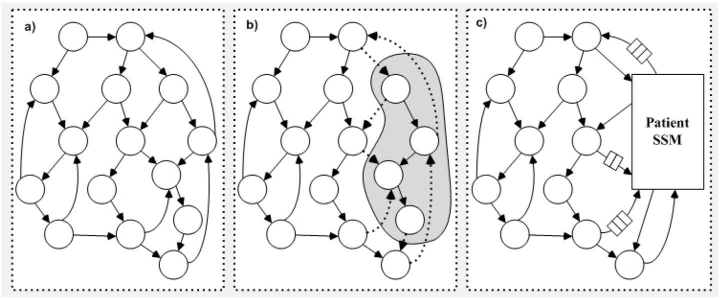

Fig. 3. a) A typical TDN graph b) The grey area is a random cut consisting of an arbitrary number of SELF components c) The selected part is transformed into a patient SSM with additional buffering at its input and output arcs to resolve hazards

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 [12]. Based on this, data-dependent computation becomes possible which means that independent data streaming can 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.

III. Synchronous Elastic Teak

From the above-mentioned features, Teak was considered as a desirable framework to investigate high-performance synchronous elasticity from a high level perspective. To incorporate synchrony in Teak, the existing component library was 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 [13]. The component set is depicted in Figure 2. 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.

IV. Synchronous Sequential Machine (SSM) Transformation

Definition 1. A SSM is a network of combinational logic such as binary gates, and sequential logic such as registers. In a SSM a cycle consisting only of combinational elements is not allowed [14]. Synchronous EDA is able to synthesise SSMs from behavioural or structural HDL specifications.

As mentioned before, the fine-grained level of elasticity in TDNs has prohibitive costs. For instance, the Sparkler processor, which is a cut-down version of the SPARC v8 architecture, consists of 2181 handshake components and 3509 links after synthesis. To reduce the handshake overhead, we developed a method to transform a partition of a TDN into an intermediate HDL representation which is a FSM extracted from CDFG with enclosed expressions within the states to realise the datapath components. Thereafter, it is possible to use synchronous EDA to synthesise and optimise the selected part. To avoid *state explosion* the method should be applied over a partition with limited number of components. For instance, the Shifter unit of Sparkler, with 27 handshake components, is transformed to a FSM with 64 states which would require further optimisations to reduce the state space.

Definition 2. A *patient* SSM is a latency-insensitive machine [15] whose registers are controlled by a global enable signal. When this signal is low, the state of the sequential elements *freezes*; no state updates occur. Any SSM is transformable into a patient SSM [14].

After generating the SSM for the corresponding partition, the machine is transformed to a patient system [14] whose clock is controlled by a synchronous elastic block [8]. According to the slack elastic property of TDN, buffering the input and output ports for any degree does not affect the functionality. We have used Synopsis' Design Compiler to synthesise the datapath elements of the SSEM processor [16] and transformed them into the patient systems. There is no reason why this method cannot be applied to more complicated synchronous machines. Figure 3 shows an abstract view of the hierarchical transformation where a random cut of a TDN is implemented as a SSM and inserted into the graph.

V. Results

As a case study the Manchester Small-Scale Experimental Machine (SSEM) is exercised [16]. SSEM is the first stored program computer in the world created at the University of Manchester in 1948. The high level specification of this computer is developed in Balsa by Andrew Bardsley and has been synthesised onto hardware using the Balsa synthesis system [17]. This machine is comprised of three separate stages which resembles commodity processors. Due to its simplicity, it is chosen as a case study to practise synchronous elasticity on a general purpose processor. Its Balsa description is synthesised using Teak and eTeak to generate asynchronous and synchronous elastic versions with the same level of granularity. In Figure 4 the area cost associated with truly asynchronous and synchronous elastic design styles is depicted. This experiment confirmed that SELF preserves slack elasticity which is the key property for our further investigations. Additionally, SELF simplifies the loop structures in the dataflow network and allows the use of synchronous CAD tools to optimise the circuit, particularly computation-heavy data manipulation units and detect the combinational loops for sake of deadlock freedom.

The following are the advantages we achieve from SELF:

- ˜4x cell area reduction
- Simplifies the loop structures regarding correctness
- Synthesise of datapath components with a standard, synchronous library

Currently, the SSEM processor needs further slack matching to improve its throughout which is under development.

At the Demo session, we show a cycle-accurate simulation of a synchronous elastic dataflow processor using the visualisation engine of Teak which allows us to trace the data tokens and the execution of the SELF protocol dynamically. Moreover, to exercise the slack elastic property of the networks, automatic buffer insertion will be available to depict the effects on the area/performance of the circuit.

VI. Conclusion and Future work

This work successfully provides a flexible communication medium based on the synchronous elastic protocol for computation in a circuit and enables exploitation of CAD tools for further transformations in the synchronous domain. Based on the analysis, elasticity at component level suffers from prohibitive costs in terms of performance as communication overhead dominates computation. A reasonable alternative is to replace these with Synchronous Sequential Machines (SSMs) and, consequently, reduce the level of elasticity. Accordingly, future work will 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.

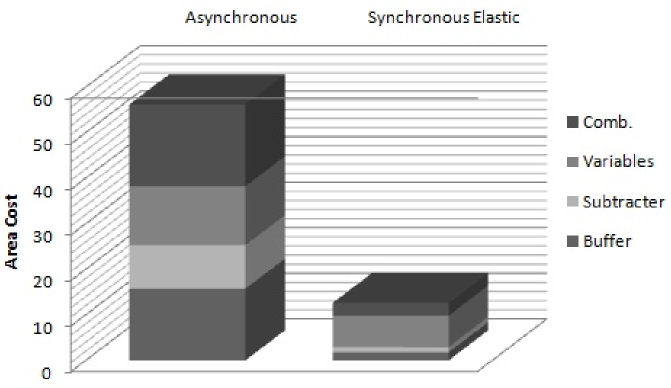

Fig. 4. The asynchronous dual-rail SSEM vs. its synchronous elastic counterpart in terms of cell area (UMC 130nm technology) which operates @ 740MHz. Each column is fragmented based on the existing entities in the circuit. Buffers are used to remove deadlock.

VII. Acknowledgment

This work is supported by EPSRC Grant “Globally Asynchronous Elastic Logic Synthesis (GAELS)” (EP/I038306/1).

References

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

[2] A. J. Martin *et al.*, “Cast: Caltech asynchronous synthesis tools,” in *Asynchronous Circuit Design Working Group Workshop, Turku, Finland*, 2004.

[3] T. Yoneda *et al.*, “High level synthesis of timed asynchronous circuits,” in *Asynchronous Circuits and Systems, ASYNC. Proceedings. 11th IEEE International Symposium on*, 2005.

[4] L. Plana *et al.*, “Attacking control overhead to improve synthesised asynchronous circuit performance,” in *Computer Design: VLSI in Computers and Processors, ICCD. Proceedings. IEEE International Conference on*, 2005.

[5] S. F. Nielsen *et al.*, “A behavioral synthesis frontend to the haste/tide design flow,” in *Asynchronous Circuits and Systems, ASYNC. 15th IEEE Symposium on*, 2009.

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

[7] A. Peeters *et al.*, “Click elements: An implementation style for data-driven compilation,” in *Asynchronous Circuits and Systems (ASYNC),IEEE Symposium on*, 2010.

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

[9] D. Sokolov *et al.*, “Gals partitioning by behavioural decoupling expressed in petri nets,” in *Asynchronous Circuits and Systems (ASYNC),IEEE Symposium on*, 2014.

[10] 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.

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

[12] Arvind *et al.*, “Annual review of computer science.” Annual Reviews Inc., 1986, ch. Dataflow Architectures.

[13] 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.

[14] M. Vijayaraghavan *et al.*, “Bounded dataflow networks and latency-insensitive circuits,” in *Proceedings of the 7th IEEE/ACM International Conference on Formal Methods and Models for Codesign*, ser. MEMOCODE, 2009.

[15] L. Carloni *et al.*, “Theory of latency-insensitive design,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2001.

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

[17] A. Bardsley, “Balsa: An asynchronous circuit synthesis system,” Master’s thesis, The University of Manchester, 1998.