eTeak: A Data-driven Synchronous Elastic Synthesiser

Mahdi Jelodari Mamaghani, Will Toms, Jim Garside
School of Computer Science
University of Manchester
Manchester, UK
{mamagham,tomsw, jdg}@cs.man.ac.uk

***Abstract*— This paper introduces eTeak, a new design flow for synthesis of the synchronous elastic systems. The proposed method inspects synchronous elasticity from asynchronous perspective and introduces a distributed control scheme on concurrent data flows specified in higher levels of abstraction. A compilation method using a new set of components is presented and future plans for this research are discussed.**

Keywords—high level synthesis; synchronous elastic flow; computer aided design

I. INTRODUCTION

Demand for more complex electronic systems with higher performance has led to aggressive scaling in semiconductor technology and it has introduced device parameter variations and consequently this effect emerges as variations in process, voltage and temperature [1] and these are responsible for systematic uncertainties in computation and communication delays. For example, the increasing ratio between global interconnect delay and gate delay complicates the design of clock trees to meet the requirements of the higher clock frequency [2]. Therefore designers are forced to cope with variability and challenges in timing closure by considering margins that could be prohibitive for the system to achieve its potential performance [3].

Robustness techniques towards variations have been widely used as advantages of asynchronous circuits [4] but the lack of mature EDA tools in this area has been the main reason why designers have been reluctant to adopt this paradigm. Recently, synchronous elasticity has emerged to exploit some of the advantages of asynchronous designs in synchronous systems to transform them to latency insensitive systems [5]. For this purpose the elastisation protocol [6] has been developed to synchronise the handshake signals with the main global clock of the system. In [3] a component set base on this protocol is introduced which are synthesisable via conventional EDA tools.

This work introduces eTeak, a high level synthesis flow for designing synchronous elastic systems. eTeak is developed based on Teak system [7] which is a token flow implementation for Balsa language [8]. It's capable of generating syntax directed data-driven handshake circuits for Balsa descriptions using a new target component set. In Teak networks datapath/control interactions are done through the local handshaking by forking/rendezvous of control and data channels. This feature enables the constructs to perform control interactions using separate 'go' and 'done' channels. The separation of go/done channels leads to fine grained concurrency and allows pipelining techniques to be applied over the generated circuits. Moreover, decoupling behaviour of Teak buffers introduces token storage over channels and it paves the road for further investigation on circuit parallelism.

In light of the mentioned features, Teak was considered as a desired framework to inspect synchronous elastic flow (SELF) from high level asynchronous prospective. In [9] a different high level method for synthesising SELF is proposed which is a control driven implementation. For reusing predesigned IP cores major modifications over the control unit should be considered. To incorporate elasticity in eTeak, the existing components are adapted to (*valid/stop*) handshaking protocol instead of (*request/acknowledge)* signalling and buffers are converted to time decoupling controllers to work based on Elastic protocol [6]. Since the new component set doesn't infer any combinational feedback loops within the network, commercial timing analysis can be used for optimisation purposes.

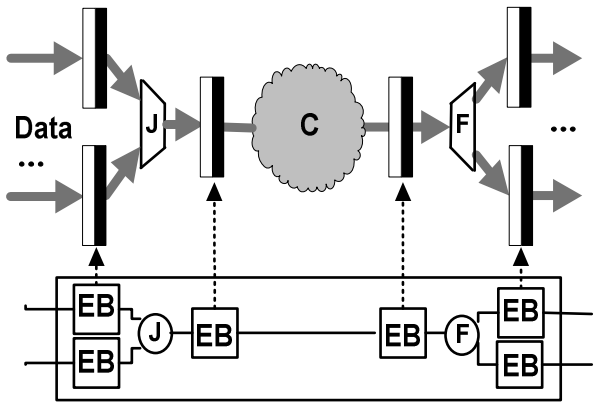


Figure 1. one stage synchronous elastic pipeline resulted from a RTL after conversion of flip-flops to EBs and insertion of J(Join) and F(Fork) components at decision points. Clock wires are not shown in the figure.

II. SELF PROTOCOL

SELF is a formalised protocol that implements a delay tolerant communication in a network. In contrast to stoppable process definition in [5], it emerges as a simple signal level handshaking between sender and receiver. The same as the conventional asynchronous protocols two wires, valid(V) and stop (S), are used to implement different states of the protocol and the components should interact in terms of these states:

- (T) Transfer, (V, ¬S): the sender provides valid data and the receiver accepts it.
- (I) Idle, (¬V): the sender does not provide valid data.
- (R) Retry, (V, S): the sender provides valid data but the receiver does not accept it.

In [6] elastic blocks (EB) are introduced and decision making Join/Fork components are described to expand the concept beyond linear pipelined communication. We can view an EB as coupled elastic half blocks (EHB) that each of them is responsible for managing an individual latch so that data tokens are decoupled on the basis of half clock cycles. In order to handle back pressure problems, EBs are considered to take control of pair of latches. In [10] a straightforward mechanism of transforming a RTL synchronous design to an elastic counterpart is described where flip-flops are simply converted to pair of latches controlled by EBs and Join/Fork components are inferred at the decision making points. In this method computational logic associated with each pipeline stage should be wrapped in between EBs (Fig. 1).

III. ELASTIC MODEL

In this section a model of elastisation is described. Fig. 2 shows a data driven structure which is synthesised from a 'forever' loop description in Balsa. This structure is the primitive model of fine grained elasticity considered by eTeak. In this model computational logic (C) communicates with the environment through channels that are controlled by EBs and Join/Forks. In this model a data token is read from a variable (V), processed by the logic and finally passed onto the successor module through a Fork (F) which simultaneously returns a 'done' to the join (J) to be able to receive the next token. The join component prevents the variable from being modified until fork returns a 'done' control token.

In order to be able to exploit this model and generate more fine grained pipelined structures 'fixed' variables must be eliminated. As mentioned in [7] retaining 'fixed' variables and mapping them directly onto hardware allows the exploration of the possible power and area implications but on the other hand this decision is not reasonable for generating pipelined structures. Accordingly, to address this problem, networks should be transformed to static single assignment (SSA) form or static token (ST) form [11] by breaking down variables into simple assignments and consequently one write/read port variables like the one shown in Fig. 2. Besides, another possibility would be to 'carry along' data and try to avoid generating unnecessary permanent storage structures. The parallel statements and parallel-written variables are the obstacles to this solution.

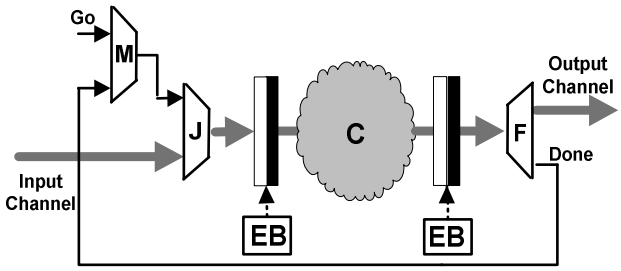


Figure 2. Our Elastisation model. A combinational logic wrapped by a pair of EBs and a chain of Merge(M)-Join(J)-Fork(F) components to control data flow. Input/output channels carry control tokens along with data tokens. Clock wires are not shown in the figure.

IV. SYNTHESIS FLOW

This section explains the synthesis flow of eTeak. The steps indicated by (*) are not complete yet and they are briefly discussed in section VI.

1) Parse tree generation: Balsa specification is parsed, expressions and commands are extracted, and the associated parse tree is generated. This step is exactly the same as Teak's.

2) Data driven network generation: each Balsa procedure is mapped onto a network part and the activation channels (go/done) are assigned and the associated input/output ports are generated in the form of channels. Since the network generation step is done in a syntax directed manner many redundant components are produced. For removing them there is a set of peephole optimisations available in [12] which is an incomplete list and still there are some to be discovered for achieving better performance.

*3) *Latch insertion:* Teak is suffering from lack of systematic latch insertion mechanism. Since the structure of the eTeak generated networks are different than the Teak's, we suspend this step until final optimisations have taken place. Balancing the pipelines for better throughput is the major target of this step.

4) Verilog netlist generation: a component set is provided to translate the network components into verilog netlist. Handshak Protocol specification and implementation is done in this step. The new component set is described in section V.

*5) *Static Ttiming Analysis* is performed on the generated verilog netlist using the conventional STA tools and critical paths are identified within the network.

*6) *Data path optimisation:*The identified paths from the previous step will be translated to behavioural verilog and synthesised using CAD tools to generate optimised structures considering target constraints.

*7) *Design space exploration*: Datapath optimisations are applicable at different levels of granularity. Morover, it is possible to explore various IP cores and consider slack matching techniques to improve the performance [13].

V. COMPONENTS

All of the seven components of Teak are modified to adapt synchronous elastic protocol instead of 4-phase

handshake signaling [2]. Therefore, the new set is greatly simplified in terms of functionality (shown in Fig. 1):

Steer – it works as a data driven de-multiplexer and Teak maps 'case' structures on this component. Each parameterised output independently matches the conditions of input. When a 'stop' happens on any of the output channels, Steer simply asserts the 'stop' for incoming tokens.

Fork – this component can be parameterised to carry any number of bits from input to output. It is frequently used for producing control 0 bit 'done' tokens from n bit data tokens. When a 'stop' happens on any of the output channels, Fork deasserts the corresponding output 'valid' signal but keeps sending data to the other available channels independently. For more details refer to [1].

Merge – this components works as a data driven multiplexer. Its service on carrying input bits to output is based on first come first served policy. So inputs should be mutually exclusive. When it gets a 'stop' signal from downstream, asserts 'stop' on every input channel to pause all incoming tokens.

Join – it's an unconditional parameterised join. It concatenates data bits of arriving inputs. A two-way join of n and 0 bits can be used as a conjunction of data and control tokens. When a 'stop' appears on its output channel, asserts 'stops' on every input channel, the same as Merge component, to pause any incoming token.

Buffer – data storage and channel decoupling. The implementation detail of this component is presented in [6]. They're responsible for synchronising valid/stop handshaking with the main global clock throughout the network. They can be inserted on channels independently to achieve any desired degree of storage.

Variable – permanent storage. Basically this component is a buffer with an extended control over its write and read activities. While 'write-go(wg)/write-done(wd)' and 'read-go(rg)/read-done(rd)' pairs make all data initializations and terminations possible, the SELF protocol adaption extremely simplifies it. In order to implement the permanent storage mechanism a 'stop' is asserted after each successful write to prevent the stored token from being modified. This 'stop' would be off whenever a read request (rg) comes up, then the variable lets the new data token be registered. Where a write done leads directly to a read go, a variable can be replaced with a simple buffer.

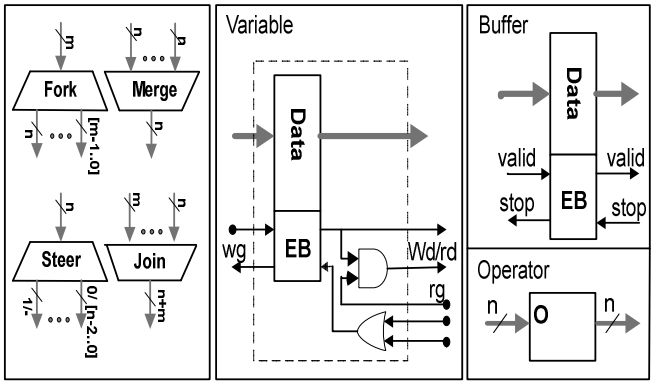


Figure 3. eTeak Components

Operator – the only components that can manipulate data. Inputs are formed into a single word through a Join. eTeak generated operators are unoptimised and this affects the performance. Therefore, further research in this area may include explorations of the efficient architectures using commercial synchronous tools.

VI. FUTURE WORK

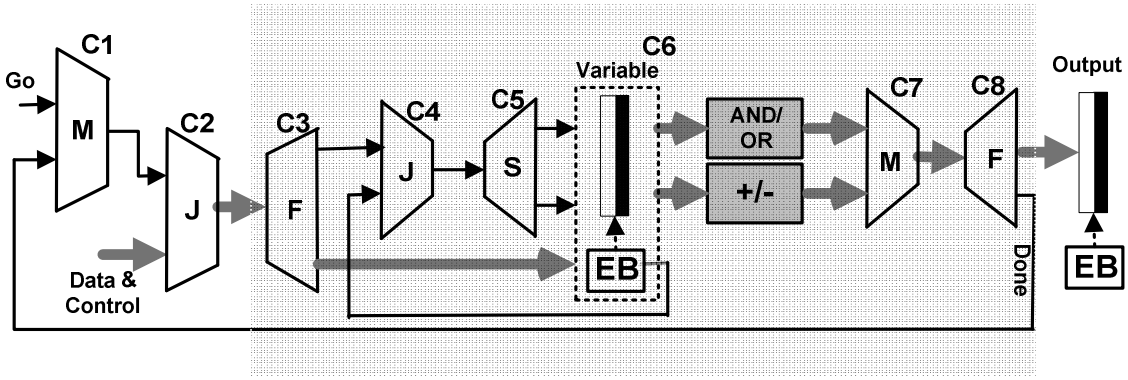
As mentioned in section IV, the syntax directed translation infers many redundant components in the network. These components are responsible for controling the data flow using asynchronous handshaking but in presence of clock signal most of them are useless and will be optimised away using commercial synchronous CAD tools. Moreover, computational blocks generated by Teak are not effiecient and it is possible to replace them with efficient IP cores.

In order to achieve our optmisation targets an interfacing mechanism between eTeak and CAD tools is required. For this purpose the associated data flow networks with identified critical paths (section IV, step 5) will be translated to behavioural verilog specifications. Then the synchronous CAD tool synthesises the code and delivers efficient strutures to eTeak to insert them back into the network by preserving the communication channels between computational units.

A simple example in Fig. 4 demonstrates how the tool is going to optimise the networks generated in primary steps. It is an ALU capable of performing logical operations, addition, and subtraction. Fiq. 4(a) shows the corresponding network generated by eTeak using the component set described in section V. In this network the combination of Merge-Join-Fork is synthesised from a 'forever' loop and is responsible for controling the flow of data as discussed in section III; C3 splits control and data tokens. Then data token is stored in variable C6 (for the sake of simplicity gate level details are not shown) and the associated control waits at C4 to ensure that data is reqistered in the variable. When C6 recives the data request, returns a control token to C4 to notify that data has been stored successfully. Then C5 recieves the contorl token and selects the corresponding path. Based on the request from C5, data token in C6 is directed through the associated path and gets manipulated by the corresponding computational block. When data arrives at C7, it forwards the token to C8 in order to retrun a 'done' token to C1 to let it know that computation is completed then C2 allows a new token to pass in.

Since the computational blocks are not optimised, timing analysis tools identifys them as potential critical paths. The grey area in Fig. 4(a) illustrates the critical path for this network and it is a candidate for further optimisations. Fig. 4(b) shows the corresponding behavioural verilog code for the critical path. Based on the structure of the network, input/output ports, data/control token widths, control points, and functional units are extracted. After synthesising the verilog code C3 to C7 will be optimised away and a combinational unit will be delivered to eTeak to wrap it by EBs and insert it back to the network.

Elastistic systems exhibit potential benefits in terms of modularity. This feature enables a wide range of exploration in



(a)

```
1       module ALU(
2        output [`max_out:0] out,
3        input  [`max_in:0] in,
4       );
5       reg [`max_out:0] out;
6       always @ (in) begin
7             case(in[`opcode-1:0])
8                `LOGICAL   : LOGIC (in[`max_in:`max_op1], in[`max_op1 - 1:`opcode], out);
9                `ADD_SUB   : ADSB (in[`max_in:`max_op1], in[`max_op1 - 1:`opcode], out);
10               Default    : out = 0;
11            endcase
12      end
13      endmodule
```

(b)

Figure 4. (a) shows an ALU, synthesised by eTeak. The grey area specifies a potential critical path identified by a STA tool. (b) shows the corresponding behavioural verilog code for the critical path generated by eTeak.

terms of power, performance, and area by utimilsing efficient predesigned IP cores. Moreover, elastisation is applicable at different levels of granularity. One option is to generate large clusters of synchrnous blocks with elastic channels between them to reduce the overhead of elasticity and exploit coarse grained voltage/frequency scaling. On the other extreme, it is possible to view the design as a set of gates. Inserting handshake at this level of granularity would be prohibitive due to the costs of large EBs.

ACKNOWLEDGMENT

This work is supported by the Engineering and Physical Science Research Council of the UK, through Grant EP/I038306/1. The authors would like to thank Andrew Bardsley for clarifications on using Teak system and Francis Southern for his useful remarks

REFERENCES

[1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and V. De, “Parameter variations and impact on circuits and microarchitecture,” in DAC, 2003, pp. 338–342.

[2] E. G. Friedman "On-chip interconnect: The past, present, and future", Proc. Netw. Chip Workshop, 2006

[3] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin. Elastic circuits. IEEE Trans. Comput.-Aid. Design 28, 10, 1437–1455, 2009.

[4] J. Sparsø and S. Furber. Principles of Asynchronous Circuit Design - a Systems Perspective. Kluwer Academic Publishers, Boston, 2001.

[5] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 20(9):1059–1076, September 2001.

[6] J. Cortadella, M. Kishinevsky, and B. Grundmann. Self: Specification and design of synchronous elastic circuits. In TAU ’06: Proceedings of the ACM/IEEE International Workshop on Timing Issues 2006.

[7] A. Bardsley, L. Tarazona and D. Edwards, Teak: A Token-Flow Implementation for the Balsa Language. In: Proc. of ACSD, 2009, 23-31

[8] Bardsley. Balsa:An Asynchronous Circuit Synthesis System. Master’s thesis (1998), Department of Computer Science, The University of Manchester,UK.

[9] G. Hoover and F. Brewer, “Synthesizing synchronous elastic flow networks,” in Design, Automation and Test in Europe, 2008. DATE ’08, 10-14 2008, pp. 306 –311.

[10] J. Cortadella, M. Kishinevsky and B. Grundmann, “Synthesis of synchronous elastic architectures” Proc. Des., Autom. Conf., pp. 657-662, 2006

[11] J. Teifel, R. Manohar, “Static tokens: Using dataflow to automate oncurrent pipeline synthesis” Proc. ASYNC, pages 17–27, Heraklion, Crete, Greece, April 2004.

[12] L. Tarazona, “Performance-oriented syntax-directed synthesis of asynchronos systems” PhD thesis (2010), School of Computer Science, University of Manchester, Manchester, UK

[13] P. Beerel , J. Cortadella and A. Kondratyev "Bridging the gap between asynchronous design and designers" *Proc. VLSI Des. Conf., pp. 18-20, 2004*