Teak: A Token Flow Implementation for Balsa

Andrew Bardsley (bardsley@cs.man.ac.uk)

School of Computer Science, The University of Manchester

Abstract

This short paper describes a new implementation method for Balsa language descriptions based only on push channels and a very small selection of components. Only a compilation method for those components is presented as this work is at an early stage.

1. Balsa

The Balsa system[1] produces Handshake Component[5] implementation of circuits described in the Balsa language. The components produced are substantially similar to those produced by the Tangram system, the precursor of Handshake Solutions' TiDE system[3].

Handshake Circuit based designs have proved to have good energy usage characteristics[4] but poor performance[6]. Both of these characteristics can be attributed to the control-heavy nature of Handshake Circuits. Explicit control channels mediate much of the data transfers between parts of a circuit. A number of methods have been used to mitigate this cost. These include the reimplementation of control circuits using petri nets and burst-mode machines[2], the replacement of critical components with more 'permissive' versions[7] and the wholesale replacement of the language, compilation method and many of the components to produce a more 'pipeline-like' implementation style[8]. Much of the optimisation work already undertaken acknowledges the role that pull channels and the push-pull role of the transferrer component have in slowing down circuits. In control resynthesis methods, the cost of preserving the original handshake enclosure between data and control handshakes in a resynthesised block of control can limit deliverable performance increases.

2. The Teak system

This paper describes a new compilation method and component set for the synthesis of Balsa descriptions. These together constitute the Teak system.

Teak replaces the data-less activation channel (used to enclose the behaviour of program fragments in Handshake Circuit compilation) with separate 'go' and 'done' channels. Control/datapath interactions using components which use signal-level event interleaving are replaced by the rendezvous/forking of control and data channels using local handshaking to complete control interactions. Explicit buffering is used to decouple one component from another and to introduce the desired degree of token storage to enable the circuit to function or, looking further ahead to more transforming compilation methods, introduce more parallelism.

Treating control channels in this way allows all the optimisation techniques usable with pipelined asynchronous systems (i.e. those with input-enclosing-output processing stages and decoupling buffering stages) to be used on Teak implemented circuits whilst still allowing local sequenced behaviour by using control channels.

3. Teak components

There are currently seven Teak components (as shown in figure 1):

Steer – conditional steer of input to exactly one output. Parameterised with disjoint match conditions for each output and bit ranges to carry to outputs. With 0 bits carried to outputs, Steer works like the Balsa 'Case' component.

Fork – unconditional n-way fork. Fork can be parameterised by which (if any) bits of the input are carried to each output. A two-way fork of n and 0 bits can be used to generate a control token from moving data.

Merge – input on one input multiplexed towards the output. Inputs must be mutually exclusive. In some configurations, Merge may have to cope with second input arrival during first input activity.

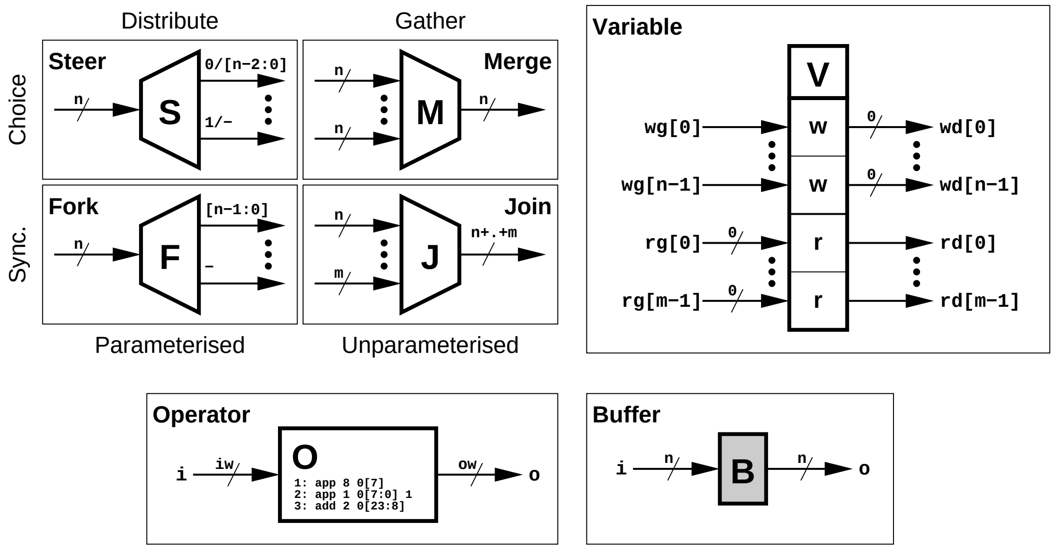


Figure 1. Teak components

Join – unconditional n-way join. Concatenates data bits of arriving inputs.

Variable – persistent storage. Separate write and read sections allow arbitrary control ordering/conditionality of reads. Variables allow complicated control activity without incurring the cost of 'carrying along' data. 'wg/wd' and 'rg/rd' (go/done) pairs make all writes data initiated and control token completed, all reads control token initiated and data delivery terminated. Where a write done leads directly to a read go, a Variable can be replaced with a Buffer.

Operator – any and all data transforming operations. Inputs are formed into a single word. Internally an Operator is organised into interconnected terms allowing Operators to be amalgamated or separated to allow cheaper implementation or Buffer insertion. The example in figure 1 is a 8 by 16 bit sign extending adder.

Buffer – data storage and channel decoupling.

All of the components, except Buffer, can be implemented with any chosen degree of input to output channel coupling (i.e. concurrency of handshaking events). Buffer must provide at least some decoupling so that it can be used to separate pipeline tokens. In this way, Teak components resemble the components of other token pipeline systems.

4. Language changes

A few Balsa language changes are necessary to make best use of this compilation style.

- Removing 'sync' channels as a concept and replacing them with 0-width types with differentiated reads and writes.
- Removing the explicit enclosure provided by the 'select' and 'chan -> then cmd end' forms of channel input command.
- The 'delivery confirmation' of sequenced channel writes provided by enclosure at 'select' and 'arbitrate' commands is no longer provided. In practice this makes control structures based on unarbitrated 'select' commands more difficult to describe.

5. Teak compilation

The Teak compilation is syntax directed, like that of Balsa. Optimisations are performed on generated component netlists. Each command in a Balsa description is mapped into components with dangling 'go' and 'done' control channels (a few commands never terminate and have no 'done'). Expressions, channel accesses and assignment lefthand sides similarly have a pair of dangling channels: one bearing data and the other a control initiating/completion channel. Control can be sequenced by joining commands ‘done’ to ‘go’ in a chain. Data and control meet with Fork and Join components.

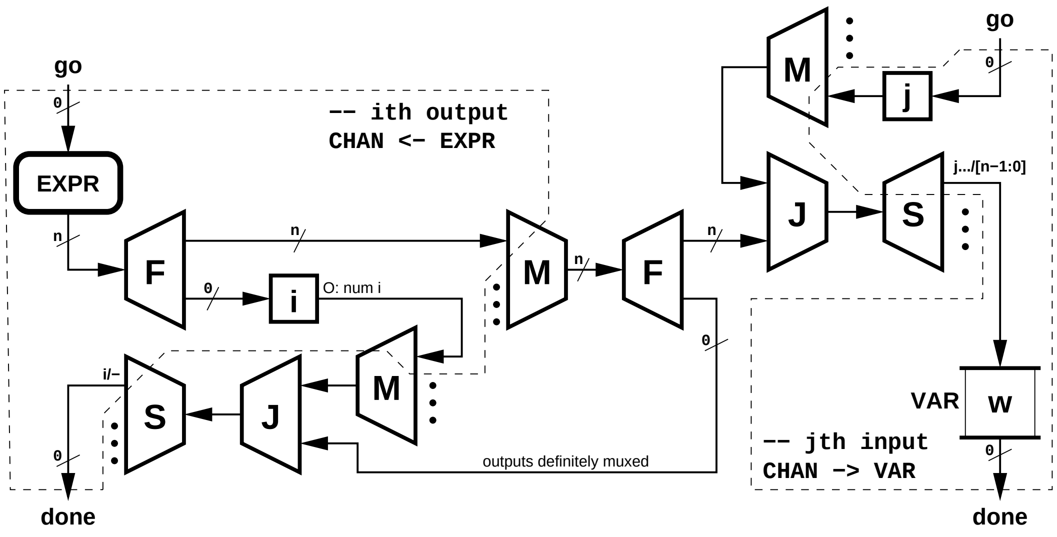


Figure 2. Language-level channel implementation

Figure 2 shows how channel read and write commands are combined to form a complete language-level channel. Note the use of Forks and Joins between data and control. In cases where ‘acknowledgement’ tokens need not be steered (i.e. where there is only one read or write to a channel in the description) much of the control/data interaction can be optimised away (as shown in figure 3).

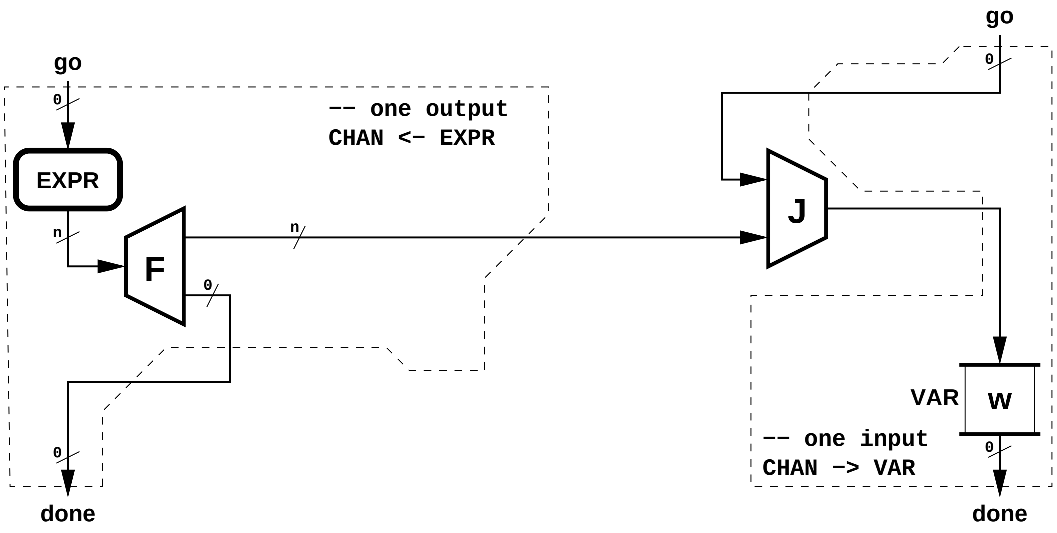
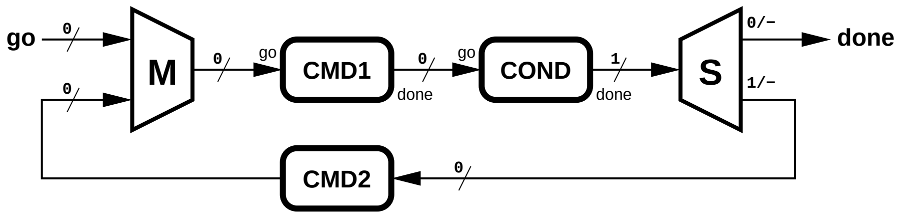


Figure 3. Channel component optimisation

Figure 4 shows the structure of a ‘while’ loop. Note that the loop containing the Merge component must have at least some buffering to prevent deadlock. Future work on Teak compilation will concentrate on where to best place such buffering.



`loop CMD1 while COND then CMD2 end`

Figure 4. While loop implementation

6. Future work

This work is at an early stage. Currently the Teak compiler can compile Balsa descriptions into either the Balsa Handshake Circuits or Teak circuits. Dual rail implementations of Teak circuits can be generated using balsa-netlist from the Balsa system. Future work includes:

- Buffer insertion schemes: fixed schemes vs. slack-matching.
- Component implementations in different data encodings.
- Lots of peephole and 'loop property' component optimisations.
- Comparisons of Teak and Balsa implementations on large demonstrators.
- A behavioural simulation and visualisation framework for Teak components.

References

[1] Balsa project homepage. URL *http://intranet.cs.man.ac.uk/apt/projects/tools/balsa/*.

[2] T. Chelcea, A. Bardsley, D. A. Edwards, S. M. Nowick. A Burst-Mode Oriented Back-End for the Balsa Synthesis System. In *Proceedings of DATE '02, pp. 330-337*.

[3] Handshake Solutions. URL *http://www.handshakesolutions.com.*

[4] H. van Gageldonk, D. Baumann, K. van Berkel, D. Gloor, A. Peeters, G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'98*. Technische Universiteit Eindhoven, Eindhoven, NL, March 1998.

[5] Kees van Berkel. *Handshake Circuits - An asynchronous architecture for VLSI programming.* Cambridge International Series on Parallel Computers 5. Cambridge University Press, 1993.

[6] L.A. Plana, P.A. Riocreux, W.J. Bainbridge, A. Bardsley, S. Temple, J.D. Garside, Z.C. Yu. SPA - A Secure Amulet Core for Smartcard Applications. *Microprocessors and Microsystems* **27(9)**, Pages 431–446 (October 2003).

[7] L.A. Plana, S. Taylor, D. Edwards. Attacking Control Overhead to Improve Synthesised Asynchronous Circuit Performance. In *Proceedings IEEE International Conference on Computer Design ICCD-2005*, pages Pages 703–710, October 2005.

[8] S. Taylor, D. Edwards, L.A. Plana. Automatic Compilation of Data-Driven Circuits. In *14th IEEE International Symposium on Asynchronous Circuits and Systems, ASYNC'08*. School of Computer Science, The University of Manchester.