Performance-Oriented Peephole Optimisation of Balsa Dual-Rail Circuits

Luis Tarazona, Doug Edwards
Advanced Processor Technologies Group, School of Computer Science
The University of Manchester, Manchester, M13 9PL, United Kingdom
Email: {tarazonl,doug}@cs.man.ac.uk

Abstract

The transparency and flexibility of the syntax-directed compilation technique, as used in the Tangram (now Haste) and Balsa synthesis systems, comes at the cost of performance overhead. Some control resynthesis and peephole optimisation techniques have been proposed to reduce this performance penalty. In this paper, we introduce new peephole optimisations for Balsa dual-rail circuits. The methods are oriented to remove redundant components and to increase the concurrency during return-to-zero phases. A novel control component, the ParSeq *(a hybrid conditional parallel/sequencer controller) is also introduced. Simulations results using large, complex design examples, show substantial increases in performance with negligible area overhead.*

1. Introduction

The syntax-directed approach used in *Tangram* [1] and *Balsa* [2] asynchronous synthesis systems, is based in the compilation of descriptions written in a high-level language into a communicating network of pre-designed modules called handshake components [3], [4]. The compilation process performs a one-to-one mapping of each language construct into a network of handshake components. The resulting network constitutes an intermediate representation that can subsequently be replaced by gate netlists. This approach gives the designer great flexibility to make changes at the description level to trade-off area, power and performance and still obtain predictable changes at circuit level. Unfortunately, this transparency and flexibility comes at the cost of significant performance overhead [4] [5].

To reduce the performance penalty of the intermediate handshake representation two main techniques has bee proposed: *control resynthesis* and *peephole optimisations*. Control resynthesis [6] is based on clustering sections of the control tree and synthesising a new controller to implement the original behaviour using a controller synthesis tool. Peephole optimisation is based on the substitution of a pattern of components with an optimised alternative using existing components. Both Tangram and Balsa use peephole optimisation as a post-processing step [4] [2]. This paper focuses in the latter technique and introduces new peephole optimisations for Balsa dual-rail circuits. The methods are oriented to remove redundant components and to increase the concurrency during return-to-zero (RTZ) phases. A novel control component, the *ParSeq* (an hybrid conditional parallel/sequencer controller) is also introduced. Simulations results using large, complex design examples, show substantial increase in performance with negligible area overhead. The optimisations targeted dual-rail encoding as this is a robust encoding scheme potentially immune to process variability that affects current fabrication technologies.

This paper is organised as follows: Section 2 presents a brief survey of the related work. Section 3 introduces the proposed optimisations. Section 4 presents the novel *ParSeq* control component. Simulation results are presented in Section 5. Finally, conclusions and future work are given in Section 6.

2. Related work

In [7], trace-theory and burst-mode-reduction is applied to improve control circuits in macromodule networks. In [8] new transform techniques for control resynthesis and peephole transforms, based on the use of a component modelling language, are presented. The optimisations were integrated with the Balsa synthesis system as a new design flow that includes an optimisation step before the synthesis steps. Datapath components are then synthesised using a Balsa burst-mode back end. Control components are synthesised using Minimalist burst-mode CAD tool [9]. Peephole optimisations falls into two classes: Protocol reversal on functional units and clustering transforms. In [4] a complete series of peephole optimisations applied in the Tangram design flow are described. The optimisations include replacement of shared datapaths, reordering and gate-level optimisations of multi-channel components. These optimisations targets both control and single-rail implementation of datapath components.

3. The Optimisations

The optimisations presented here use template-based matching to identify and substitute a component or a cluster of components with an optimised version. In some cases, the substitution eliminates redundant components, resulting in both area and speed improvements. These optimisation include: elimination of redundant single-port false variables, utilisation of concurrent RTZ Fetch component, and use of the *ParSeq* component. This last optimisation will be discussed in section 4

3.1. Eliminating redundant single-port false variables

Active input control can be used in Balsa when there is no input choice. In this case, a *Fetch* (`->`) and a *FalseVariable* (FV) component are used to implement this construct. Figure 1 shows a single channel active input control. In cases when the channel is read only once and the control simply transfers the value to a consumer module in the datapath, the

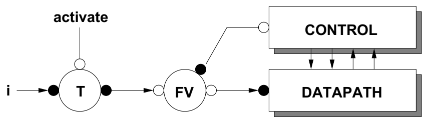

Figure 1: Active input control.

FalseVariable can be removed safely. This can also be done with the *Eager FalseVariable* component. As an example, consider the Balsa code fragment of figure 2 where two input channels, `a` and `b`, are read and then added to produce the output `o`, which is of type `outType`. Note that the reading of the input channels encloses the output operation.

```
     ⋮
a,b -> then
  o <- (a + b as outType)
end
     ⋮
```

Figure 2: Fragment of Balsa code showing single-read active input.

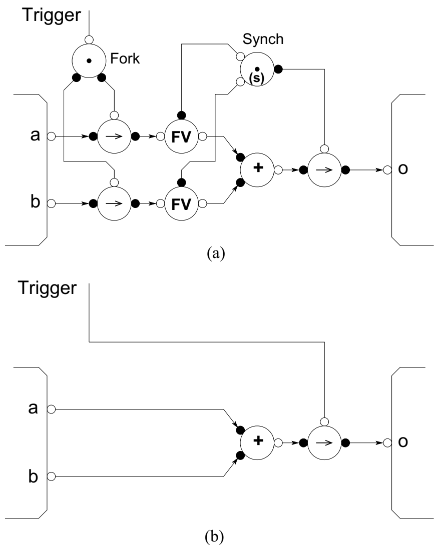

Figure 3: Handshake circuit for example in figure 2, (a) orginal, (b) optimised.

The resulting handshake circuit is shown in figure 3(a), where the trigger signal comes from the control that initiates the transfers. The *Synch* (s) component is used here to initiate the transferring of the output only after the first data bits of a and b arrive, as described elsewhere [5]. In this case, both the *FV* and the *Synch* are redundant: the control can initiate the reading operation by triggering the transferrer at the output, which almost immediately starts the pulling of the values from the input channels. Figure 3(b) shows the resulting circuit. This transformation obviously results in latency and area reduction, yet preserving the external behaviour of the circuit.

3.2. Utilisation of concurrent RTZ Fetch component

The Balsa dual-rail Fetch component, shown in figure 4(a), consists of wires only, with broad data validity in both data ports. The signal transition graph (STG) in figure 4(b) shows how the RTZ phases of the activation, input and output are fully sequenced. In [4], Peeters described two single-rail transferrers with concurrency in the data channels, the *par-ser* and the *ser-par*, but its implementation in dual-rail would require completion detection inside the Fetch.The Fetch proposed here, focuses on the concurrency of RTZ phase of data and activation channels: if the activation channel depends on the RTZ of a wide data channel, the completion detection of that channel will delay unnecesarily the RTZ of the input data channel. In Balsa circuits, this situation occurs when the activation is generated by a *Case* component whose input data port has a considerable number of bits, as found in the implementation of the write index for array variables with many entries.

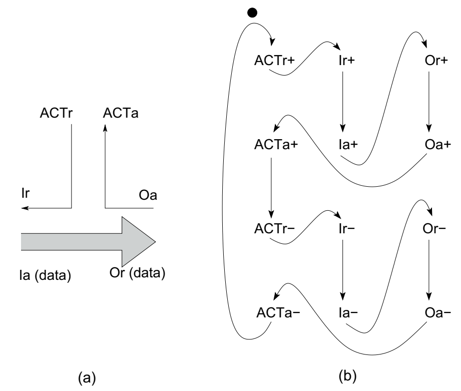

Figure 4: (a) Conventional Balsa dual-rail Fetch. (b) Its STG.

A more concurrent operation can allow the RTZ on the activation channel in parallel with the RTZ on the data channels. Figure 5 (a) shows the new circuit for the dual-rail fetch and the resulting STG. Substitution of the wires-only Fetch should only be made when the Case component that activates it has a slower RTZ than the delays introduced by the controller in the new Fetch, but this threshold is easily tunable.

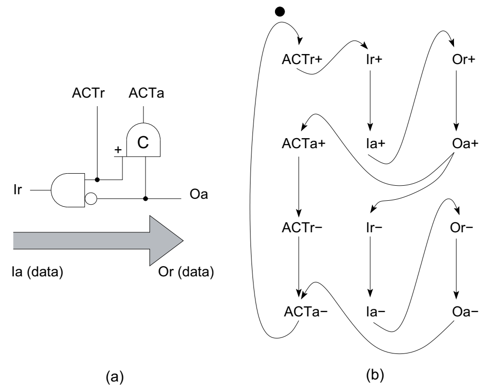

Figure 5: Fetch with concurrent RTZ (a) and its STG (b).

4. The *ParSeq*

The *Concur* (||) and the *Sequencer* (;) are control components used to implement parallel or sequential composition of commands. The idea of a component that can conditionally act as a *Concur* or as a *Sequencer*, when it is safe to do so, led to the implementation of the *ParSeq* (from *Par*allel and *Seq*uencer) component. Situations where a *ParSeq* can be used were very few in the descriptions analysed: the general purpose RISC processors nanoSpa [10] [11] and SAMIPS [12]. A possible reason for this could be that, as the operator was not available, the description cannot favour the use of *ParSeq*. In spite of this, the use of a single *ParSeq* allowed an interesting increase in performance, as will be shown in section 5.

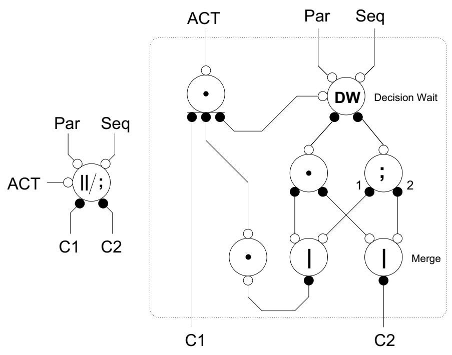

Figure 6: Symbol and a handshake circuit implementation of the *ParSeq* component.

Figure 6 shows an implementation of the *ParSeq* using existing Balsa HCs. Figure 7 shows the schematics of the optimised implementation. The boxes labelled T/S are either T-elements or S-elements according to the level of concurrency allowed for safe sequential operation [5]. The component has one activation port (*ACT*), two mutually exclusive passive nonput input ports (*Par* and *Seq*) to indicate the desired behaviour, and two nonput active output ports (*C1* and *C2*). The operation is as follows: A request in the *ACT* channel enables the operation and a request on either *Par* or *Seq* (that can occurr concurrently with *ACT*) will generate either concurrent handshakes in *C1* and *C2* or (if *Seq* is activated) a handshake in *C1* sequenced by a handshake in *C2*. Handshakes in *C1* and *C2* are enclosed by handshakes in *ACT* and *Par* or *Seq*. In situations where there no exist write-after-write (WAW) hazards between the sequenced operations, the C-element surrounded by a dashed box in figure 7, can be replaced by an AND gate for concurrent RTZ phases on the C2 and ACT channels. Figure 8 shows the STG of the *ParSeq* using T-elements. For clarity, the arcs corresponding to parallel, sequential and unconditional behaviour have different colouring and width. The environment must ensure the mutual exclusivity between *Par* and *Seq*. In general, the evaluation of the conditional value is done using a *Case* component that ensures the required mutual exclusion.

5. Results

Table 1 shows the results of the proposed optimisations when applied to the nanoSpa 32-bit processor, using pre-layout, transistor-level simulation in 180nm technology.

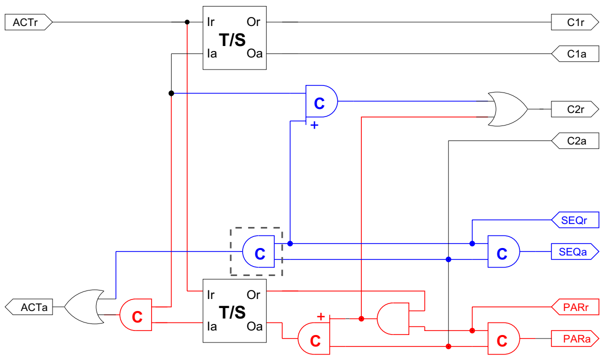

Figure 7: *ParSeq* module schematic circuit.

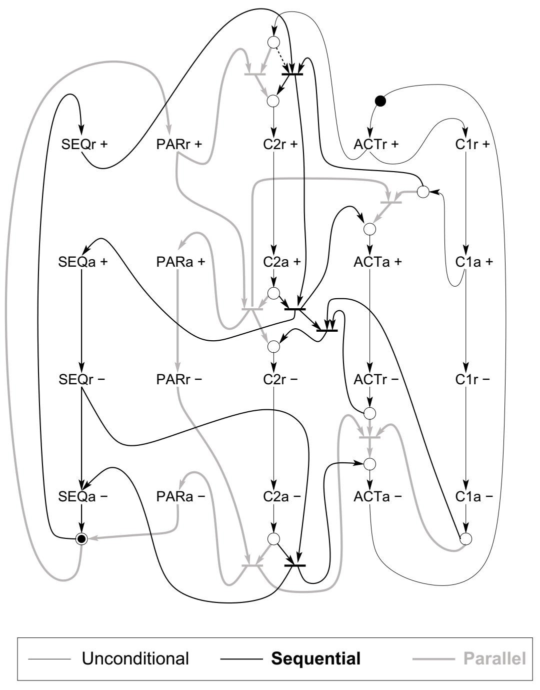

Figure 8: *ParSeq* component STG.

Results show that the removal of redundant FVs increases the performance up to 2.6% and reduces the area by a negligible amount. The use of new concurrent Fetch component gives close to 4% improvement in performance at practically not cost in area. The use of *ParSeq* gives close to 5% performance improvement with less than 3% area increase. The reasons for this area increase will be given in the next paragraph. When all optimisations were applied, the overall increase in performance was more than 10% with less than 3% area penalty. Table 2 compares the results for the *ParSeq* implemented using existing HCs versus the optimised circuit in nanoSpa and SAMIPS processors. The table shows that there is a performance penalty of 2.27% when the *ParSeq* component is implemented using existing HCs, whereas the use of the optimised module shown in figure 7, with the C-element replaced by an AND gate, gives up to 5% performance improvement.

Regarding the use of the *ParSeq*, the following observations must be made:

In the descriptions of the processors used as examples, the

Optimisation	DMIPS	perf. gain %	area ov. %
none	73.54	0.00	0.00
removal of redundant FVs	75.43	2.60	-0.11
new Fetch	76.23	3.66	0.25
ParSeq ”T”	77.07	4.79	2.51
All of above	81.01	10.15	2.65

Table 1: Effects of optimisations when applied to the nanoSpa processor.

implementation of the sequenced reads and writes of the register bank was the most interesting place in the descriptions were the *ParSeq* can be used: sometimes the writing process has nothing to write, so it can proceed concurrently with the read. As this operation does not introduce WAW hazards, the more concurrent version of the component was used. Another importan observation is that, also for performance reasons, a *ParSeq* with T-elements was used. As discussed in [5], this can potentially lead to write-after-read (WAR) hazards. To avoid this, a modified version of the *Variable* component, that can be safely written concurrently while reading, was used. The storage cells of this version consist of a master-slave latch where the slave latch is controlled by the read signal, so overwriting will occur only after the read signal is deasserted. The almost double size of the safe variables are the source of the area increase shown in table 1 when using the *ParSeq*. implementation.

Device	Optimisation	perf. gain %	area ov. %
nanoSPA	*ParSeq* ”T”	4.79	2.51
nanoSPA	*ParSeq* ”T” HC	-2.27	2.51
SAMIPS	*ParSeq* ”T”	3.94	<0.01
SAMIPS	*ParSeq* ”T” HC	-1.31	<0.01

Table 2: Performance and area simulation results of *ParSeq* when used in nanoSpa and SAMIPS.

6. Conclusions and future work

The work presented in this paper demonstrates that the optimisations proposed for Balsa dual-rail circuits can achieve more than 10% of improvement in performance, at practically no area cost, when applied to large, complex designs. This paper also introduced the *ParSeq*, a novel control component that allows the conditional parallel or sequential composition of commands. Results using pre-layout, transistor-level simulation in a 180 nm technology, show that the *ParSeq* can achieve non-trivial performance improvements.

Future work includes incorporating this optimisations in the Balsa design flow. At the moment it has not been decided yet if the *ParSeq* will be incorporated as an optimisation or as a new operator. Further work will also include examining other potential peephole and handshake components optimisations.

Acknowledgment

The authors would like to thank Luis Plana, Charles Brej, Andrew Bardsley and William Toms, all of them members of the Advanced Processor Technologies group, for all their cooperation, suggestions and comments during the development of this work.

References

[1] Kees van Berkel, Joep Kessels, Marly Roncken, Ronald Saeijs, and Frits Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proc. European Conference on Design Automation (EDAC)*, pages 384–389, 1991.

[2] A. Bardsley. *Implementing Balsa Handshake Circuits*. PhD thesis, Department of Computer Science, University of Manchester, 2000.

[3] Kees van Berkel. *Handshake Circuits: an Asynchronous Architecture for VLSI Programming*. Cambridge University Press, 1993.

[4] A. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven University of Technology, June 1996.

[5] Luis A. Plana, Sam Taylor, and Doug Edwards. Attacking control overhead to improve synthesised asynchronous circuit performance. In *Proc. International Conf. Computer Design (ICCD)*, pages 703–710. IEEE Computer Society Press, October 2005.

[6] Tilman Kolks, Steven Vercauteren, and Bill Lin. Control resynthesis for control-dominated asynchronous designs. In *Proc. International Symposium on Asynchronous Circuits and Systems*, pages 233–243. IEEE Computer Society Press, March 1996.

[7] Ganesh Gopalakrishnan, Prabhakar Kudva, and Erik Brunvand. Peephole optimization of asynchronous macromodule networks. *IEEE Transactions on VLSI Systems*, 7(1):30–37, March 1999.

[8] Tiberiu Chelcea and Steven M. Nowick. Resynthesis and peephole transformations for the optimization of large-scale asynchronous systems. In *Proc. ACM/IEEE Design Automation Conference*, June 2002.

[9] R. M. Fuhrer, S. M. Nowick, M. Theobald, N. K. Jha, and L. A. Plana. MINIMALIST: An environment for the synthesis and verification of burst-mode asynchronous machines. In *Proc. International Workshop on Logic Synthesis*, June 1998.

[10] L.A. Plana, D. Edwards, S. Taylor, L. Tarazona, and A. Bardsley. Performance-driven syntax directed synthesis of asynchronous processors. In *Proc. International Conference on Compiles, Architecture & Synthesis for Embedded Systems*, pages 43–47, September 2007.

[11] L. A. Tarazona, L. A. Plana, and D. A. Edwards. Architecture enhancements for a synthesised self-timed processor. In *Proceedings of the UK Asynchronous Forum*, September 2007.

[12] Q.Y. Zhang and G. Theodoropoulos. Towards an asynchronous MIPS processor. In *Cryptographic Hardware and Embedded Systems (CHES 2003)*, volume 2779 of *Lecture Notes in Computer Science*, pages 137–150. Springer-Verlag, 2003.