# The Balsa Asynchronous Circuit Synthesis System

A. Bardsley, D. A. Edwards

*Department of Computer Science, The University of Manchester*
*Oxford Rd., Manchester, M13 9PL, United Kingdom*
*bardsley@cs.man.ac.uk, doug@cs.man.ac.uk*

## Abstract

*The Balsa synthesis system is presented. Balsa generates purely asynchronous macromodular circuits from CSP-like descriptions similar to those of Philips' Tangram tool. Balsa targets standard-cell and FPGA technologies by producing gate level netlists. Place and route can be performed with existing commercial tools. The DMA controller for the AMULET3i asynchronous microprocessor macrocell is presented as a proof of design flow. AMULET3i is to be fabricated in 0.35 µm 3LM CMOS. Silicon is due to be received in August 2000.*

## 1. Introduction

This paper describes a synthesis system created to explore the use of asynchronous logic design to implement channel orientated descriptions of hardware. The system is modelled after the Philips Tangram HDL and tools [15] which have been successfully used to implement a number of ICs, most notably the Myna Pager [13] which includes a fully asynchronous implementation of the Intel 80C51 microcontroller [11]. Tangram introduced the concept of *handshake components* and *handshake circuits* [5] and has done much to popularise the use of asynchronous circuits as a synthesis target technology.

### 1.1. Asynchronous design

Early computers were designed using asynchronous techniques. However the ad-hoc nature of the design process, particularly delay management, and the difficulty in ensuring race-free and hazard-free operation led to the near universal adoption of a synchronous style of operation. Here, a global clock provides a means of abstracting away functional issues from those of time and delay management.

In the last 10 years, fundamental research has solved many of the issues of designing asynchronous finite state machines (AFSMs) and new methodologies have provided a systematic way of constructing large systems. Proponents of asynchronous technology have claimed the following advantages:

- better EMC
- low power operation
- higher performance (average-case rather than worst-case)
- the avoidance of clock distribution problems
- modularity

This renewed interest has led to the development of new tools to tackle the problem of automating new asynchronous techniques and making them more palatable to synchronous designers. These include:

- the 'NULL Convention Logic' system from Theseus Logic [17]
- the Petrify [6] petri-net based AFSM design tool
- the Minimalist [8] burst mode machine based AFSM design tool
- The Philips' Tangram hardware description language mentioned above.

It is clear, from interest being shown in asynchronous design start-ups, that asynchronous technologies are now gaining commercial credibility in several significant niche areas:

#### 1.1.1. Low EMI

The low, uncorrelated EMI (Electro-Magnetic Interference) generated by asynchronous circuits allows applications not possible in equivalent clocked designs. In the Philips Myna pager, the low EMI produced by the asynchronous microcontroller allows the processor to be left active dur-

ing the reception of radio traffic. Consequently, the three current pager standards can be accommodated in software rather than requiring different hardware for each standard which would have been the case if a clocked system had been employed. A generic design can thus be used for all standards thereby lowering overall design and production costs.

#### 1.1.2. Clock-Skew-Free Design

Sharp have announced a self-timed data-driven media processor having a peak performance of 14.4 $GOPs^{-1}$ [16]. Sharp cite the ability to build elastic pipelines without worrying about clock-skew as a major contributer to the high performance of their product.

The examples outlined above are concrete instances of the value of asynchronous technologies. It is probable that other niche areas will emerge. A-D converters can have enhanced performance and reliability [14] when built using self-timed circuits. Asynchronous macrocell buses [4] have many advantages over their synchronous counterparts allowing macrocells with differing intrinsic clock frequencies to be interfaced easily.

#### 1.1.3. I.P. (intellectual property) reuse

The modularity offered by self-timed systems facilitates design reuse. The use of internal delay-signalling in many asynchronous circuits produces designs which can be placed and routed in larger designs without compromising timing closure. In this way, asynchronous I.P. can be offered as both hard (pre-placed and routed) and soft (unplaced/synthesisable) macrocells. Delay-insensitivity can also make reimplementation due to cell library migration and process feature size reduction easier.

#### 1.1.4. Zero power idle operation

Uniquely, asynchronous systems can be put into and out of a zero-power sleep mode very rapidly (sub-ms time) [1]. AT&T Laboratories (formerly Olivetti Research Laboratory) are exploiting this property in their Piconet [2] short-range ubiquitous wireless LAN system using the AMULET2e [9] processor. The receiver can sleep (thus saving power) even between packets sent over the radio link. Philips have also demonstrated significant power savings in their asynchronous implementation of the DCC error corrector [12].

### 1.2. Handshake circuits and components

Handshake components are parameterisable components used as an implementation technology independent intermediate for synthesis in a similar manner to the EDIF LPM component set [7]. Unlike the EDIF component set, each of the terminals of the handshake components is accompanied by request/acknowledge signalling to indicate when the data on that terminal is ready and when the data has been accepted by the party connected to that terminal. In this way handshake components communicate solely by taking part in data handshakes and are connected together solely by channels including this handshake signalling.

The use of cooperative handshaking in handshake circuits (the compositions of handshake components) allows circuits to be built which do not require a global clock for internal synchronisation. A handshake circuit thereby presents a very flexible, modular interface to the world. Handshake circuits can also be themselves partitioned into modular subcircuits (to map onto a number of ICs/FPGAs for instance) by separating the circuits' components into groups and using the channels connecting those groups as the interfaces between those groups. Figure 1 shows how two handshake components are connected by a channel. Here, the connection is between a Fetch component and a Case component (the use of this component combination will be explained in the compilation example below). The Fetch component presents a handshake requests and data to the Case component using an 'active' port (with a filled circle) which the Case receives on a 'passive' port. Data follows the direction of the request in this example and the acknowledgement to that requests flows in the opposite direction. In this figure, individual, physical request/acknowledgement and data wires are explicitly shown. Data is carried on separate wires from the signalling (it is 'bundled' with the control) although this is not necessarily the case with other data/signalling encoding schemes. Normally, handshake circuit diagrams show the channel as a single arc where the control and data directions can be descerned from the passive/active nature of the ports connected and an arrowhead on the arc indicating data direction.

Methodologies exist (DI codes, dual rail encoding, NULL Convention Logic) to implement channel connections with 'delay-insensitive' signalling where the timing relationships between individual wires of an implemented channel do not affect the functionality of the channel. The methodologies can be used to implement handshake circuits

which are robust to naive implementation, process variations and interconnect delay properties. In their interfaces to other asynchronous circuits, handshake circuits usually match data rates by the use of request/acknowledge signalling, no (potentially dangerous) explicit synchronisation between clock domains is required. Where asynchronous circuits are interfaced to synchronous circuits, the problems of synchronisation are no worse than between different synchronous clock domains.

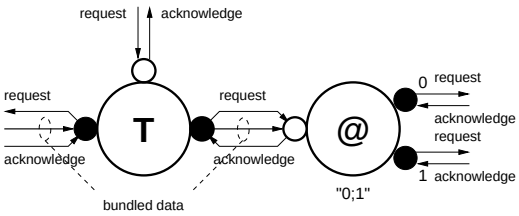


**Figure 1. Two connected handshake components**

## 2. The Balsa system

Balsa departs from Tangram in promoting a more coarse grain approach to handshake circuits, promoting the use of components which are parameterisable by behaviour and by aspects such as terminal sizing and numbering. The aim of Balsa is to provide syntax directed compilation without the need to gate-level optimise a flat netlist in order to produce suitably small and fast circuits. In the design of asynchronous circuits, gates are often added which a gate-level optimiser may view as superfluous. These include gates whose purpose is to provide protection from hazards present in control circuits and to monitor key control signals (such as latch enable signals) to provide indications that operations have been completed. Producing larger, more parameterisable handshake components instead of clusters of smaller components allows such optimisations to either be performed at a higher level, by careful implementation for example. Peephole optimisation can be performed in a restricted manner inside those components with their interconnections being exactly as described in the original syntax-directed-compiled description.

An overview of the Balsa design flow can be found in figure 2. This flow shows the use of LARD to perform behavioural simulations but the target CAD system (after balsa-netlist) can also be used to perform more timing-realistic simulations and to validate the design. Most of the Balsa tools are concerned with manipulating the Breeze handshake intermediate files. Breeze files can be read by back-end tools to derive implementations for Balsa descriptions but also contain procedure and type definitions passed on from Balsa source files allowing Breeze to be used as the package description format for Balsa.

Improvement of the area/performance characteristics of a Balsa design is usually performed by modifying the Balsa description for a circuit and then testing the effect of that modification in simulation. Using syntax-directed compilation allows the designer a clear understanding of the effect of source description modifications on the implementation making the process of design refinement by description rewriting much simpler than for less transparent synthesis mechanisms.

Design area estimation is possible using the breeze-cost tool (not shown on figure 2) on a Breeze netlist file. Design refinement can be performed using this area estimate and the (very) rough timings available from LARD behavioral simulation allowing design refinements to be evaluated independent of a target implementation technology.

Timing validation of Balsa generated implementations is not yet implemented as part of the Balsa design flow. This form of validation is currently undertaken through simulation although the use of existing synchronous static timing analysis tools is to be investigated as part of further development of Balsa.

A small example of Balsa code is given below. This is a modulo-10 counter described with a two port external interface: port `aclk` is a dataless port which provides external stimuli causing the counter to increment and port `count` is an output onto which the counter places that value each time a handshake is performed on `aclk`. The counter is defined as a Balsa procedure which can subsequently be composed with other procedures to form an entire design. As the ports to a Balsa procedure are channel connections, the calling mechanism for such a procedure involves activating the counter and then providing stimulus to the procedure's ports. This is similar to the behaviour of VHDL processes except that Balsa procedures may be composed sequentially (for one-shot operation, to provide distributed access a shared resource perhaps) or in parallel (as permanent components) as may any two, non-resource-conflicting, commands in the language. The body of procedure `mod10` is wrapped in a '`loop … end`' infinite loop command giving this procedure a repeated behaviour very much like that of a VHDL process.

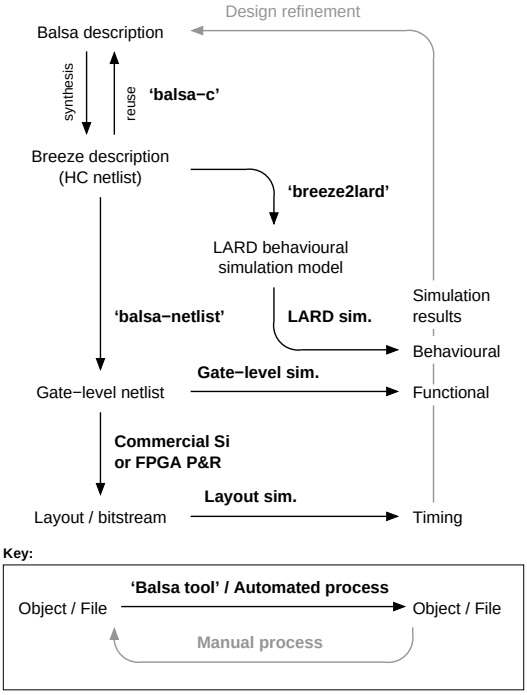


**Figure 2. The Balsa design flow**

```
-- mod10.balsa: async.
-- decade counter

import [balsa.types.basic]
public

type C_size is nibble
constant max_count = 9

procedure mod10 (sync aclk;
  output count : C_size) is
local
  variable count_reg : C_size
  variable tmp : C_size
begin
 loop
  select aclk then
    if count_reg /= max_count then
      tmp := (count_reg+1 as C_size)
    else
      tmp := 0
    end;
    count <- count_reg;
    count_reg := tmp
  end -- complete select H/S
 end -- loop end
end
```

The 'select … then … end' command encloses the behaviour of the procedure body inside the handshake on aclk and the output command 'count <- count_reg' makes the count value available on the port count. Note that the output command is not equivalent to a signal assignment is languages such as VHDL or Verilog, it is a channel communication and actively participates in a handshake with the component connected to the port count (or, in this case, the test harness) to transfer the value in count_reg to that component. In this way the communication is both a data moving operation and a control feature in its own right. This method of typing together processes in the design description using communicating 'synchronous' channels (i.e. the sender and receiver are synchronised by the communication) is similar to CSP. Balsa extended the CSP communications semantics, however, to include the ability to hold a channel communication open while a command is executed. This behaviour is implemented by the 'select' command.

Figure 3 shows an example handshake circuit. This is the modulo-10 counter whose Balsa description is given in the next section. The counter has three ports when implemented: activate, aclk and count. The activate port is used to reset the circuit and provide the initial event which starts the circuit's behaviour. The other two ports' functions are described below. This circuit consists of 9 types of component: Loop, DecisionWait, Sequence, BinaryConstFuncR, Fetch, Variable, Case, Constant and CallMux.

DecisionWait, Loop and Sequence (shown by the 'DW', '#' and ';' symbols in the large circles which represent HCs) provide the overall circuit control and sequence the actions of the datapath. The data processing component BinaryFuncConstR appears twice, once as 'x /= 9' and once as 'x + 1' BinaryFuncConstR implements all the binary datapath operations which involve one constant and one variable input. Data is stored within the circuit in latch components called Variables (the 'tmp' and 'count_reg' components to match the variable names found in the Balsa description). Variables have a single write port with control signalling activating the latch enable and a number of read ports with data/control delay matching.

Where datapath and control components meet, a Fetch component ('T') is used to transfer data from one place to another. Fetch has three ports: a data input, a data output, and a control port. Handshakes on the control port cause a request-for-input on the data input port. On receipt of input data, that data value is output on the Fetch's output port. Fetch components are typically implements by just cross connecting the control signals between the three ports and feeding through data making them effectively free components. The final component is Case, Case (symbol '@') implements a case or

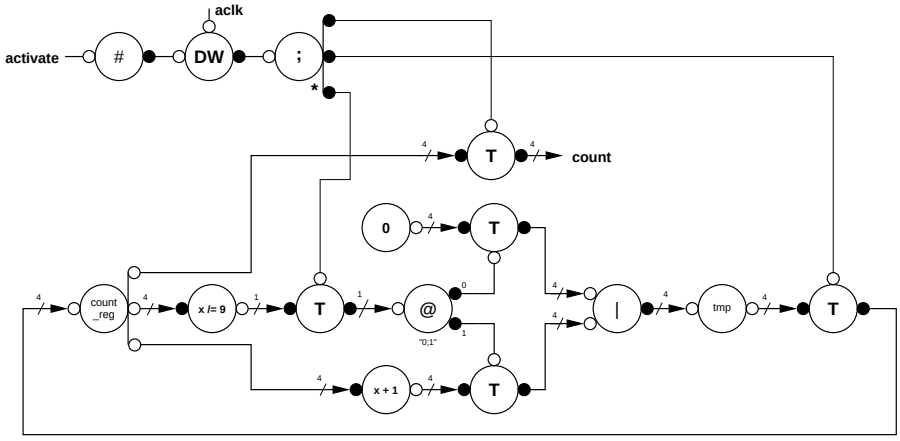


**Figure 3. Modulo-10 counter example – HC level**

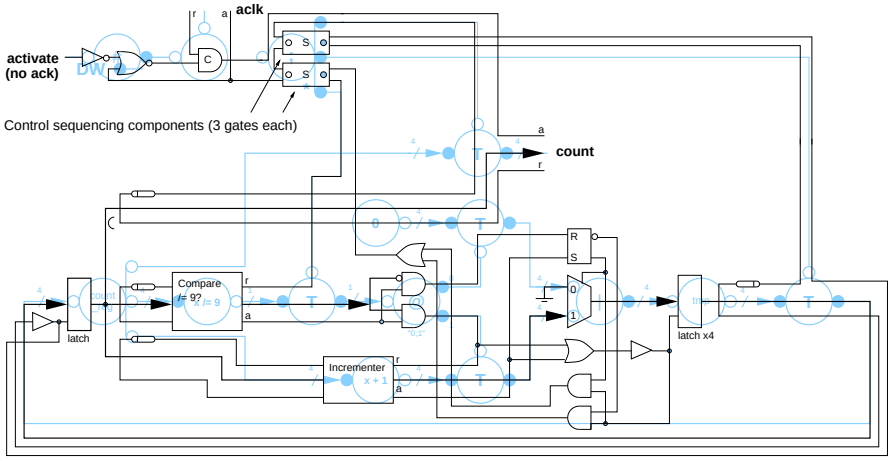


**Figure 4. Modulo-10 counter example – gate level**

`if` command control choice. Here the Case component directs either a constant 0 (from a Constant component) or the result of adding 1 to the variable count_reg into the variable tmp. Comparing this circuit with the code given below, it should be obvious that this is a direct implementation of that circuit description.

Figure 4 shows the gate level implementation for mod10. This is a pre-optimised netlist and the direct mapping of handshake components to gates can be clearly seen. After gate level peephole optimisation, this circuit becomes somewhat smaller. The multiplexers with constant values on their inputs can be reduced to AND gates, most of the delay matching elements, shown as lozenges, are removed and the decoder/encoder pair of the Case and Call components can be simplified. Future development of Balsa will concentrate on removing the need for gate level optimisation by providing a richer set of more configurable handshake components to allow HC peephole optimisation to replace gate level optimisation. Reducing the reliance on flat gate level circuit optimisations helps avoid over-eager optimisation which can ruin the properties of asynchronous circuits.

An example simulation of this counter is shown in figure 7. The simulation shows the counter being repeatedly incremented by a simple test bench. The left hand window shows channel activity with the activity on port count obviously concurrent and enclosed within the handshake on `aclk`. The pairs of blocks for each channel indicate request and acknowledge, values on channels are shown in small white boxes. The right hand window is the source level Balsa debugger and is indicating that the procedure mod10 is waiting for input on channel `aclk` in thread number 8.

This example is presented as an illustration of the Balsa language and its compilation flow. For familiarity, a well known synchronous style counter has been used. Experienced asynchronous designers would probably use a counter built in a style more appropriate to their asynchronous implementation methodology for example one of the constant response time systolic counters described by van Berkel [3].

## 3. The AMULET3i DMA controller

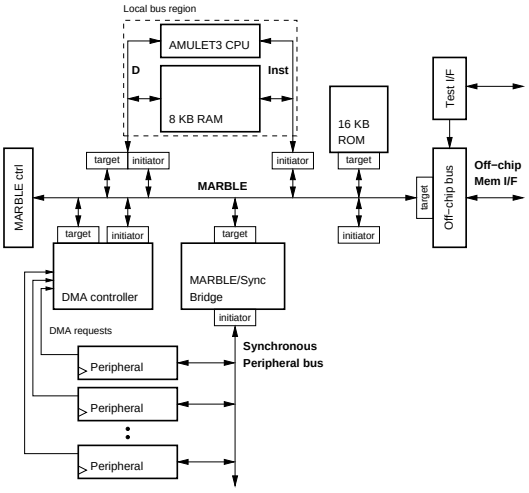


**Figure 5. AMULET3i macrocell and synchronous peripherals**

The AMULET3i DMA controller is presented here as a substantial example of the practical application of Balsa to a real-world design problem. Balsa was used to implement the controller due to the uncertain state of the design specification at the time at which the AMULET3i project was undertaken because Balsa promised to allow faster design revision turn-around in response to specification changes.

The AMULET3 [10] is a third generation fully asynchronous microprocessor developed by the AMULET Group at The University of Manchester to form part of an asynchronous macrocell suitable for use in SoC applications. AMULET3i (shown in figure 5) is that macrocell and consists of: an AMULET3 processor, 8 KB of static RAM, 16 KB of mask programmable ROM, off chip memory interface, simple synchronous peripheral bus interface and a DMA controller. The components of the macrocell are tied together using an asynchronous bus called MARBLE [4]. The DMA controller is required to perform transfers between the AMULET3i macrocell and customer/integrator provided synchronous peripherals on the far side of the MARBLE/synchronous bus bridge. In order to provide bus master capabilities not provided by the simple synchronous bus, the DMA controller is situated on the MARBLE side of bridge and presents both its programming (target) and bus transfer (initiator) interfaces to MARBLE.

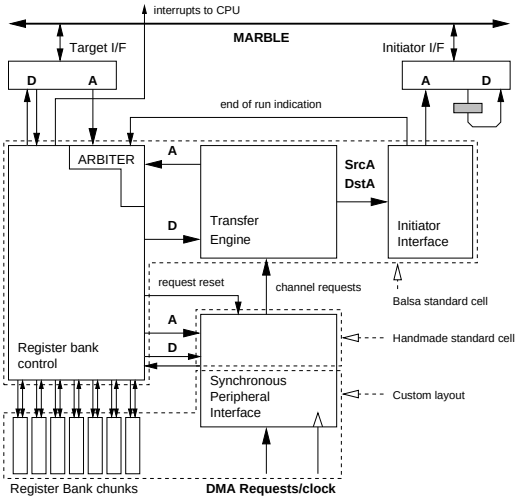


**Figure 6. AMULET3i DMA controller structure**

The design of the DMA controller was greatly influenced by the changing needs of the first intended customer for AMULET3i and as such has undergone many design revisions during its development. Balsa was used to implement the controller because of its fast design turn around time, typical of synthesis systems, and in order to test the Balsa system in a real-world application. The DMA controller is connected for its 'DMA request' stimuli to each of the synchronous peripherals (shown in figure 5 as the thin lines) and performs transfers based on the status of these lines. Figure 6 shows the internal structure of the DMA controller and its partitioning between synthesised Balsa, hand designed standard cell and custom layout.

The arrival of a DMA request at the block marked Synchronous Peripheral Interface triggers a transfer by the Transfer Engine and Initiator Interface on the MARBLE initiator bus interface. Transfer use programmed addresses and transfer counts held in the Register bank control and the full-custom register bank chunks. The DMA controller provides support for up to 16 peripherals with 32 transfer channels (transfers may be 'cascaded' between channels on a single request making this choice of channel numbers not so unusual), only 6 of these channels support full 32 b addressing and

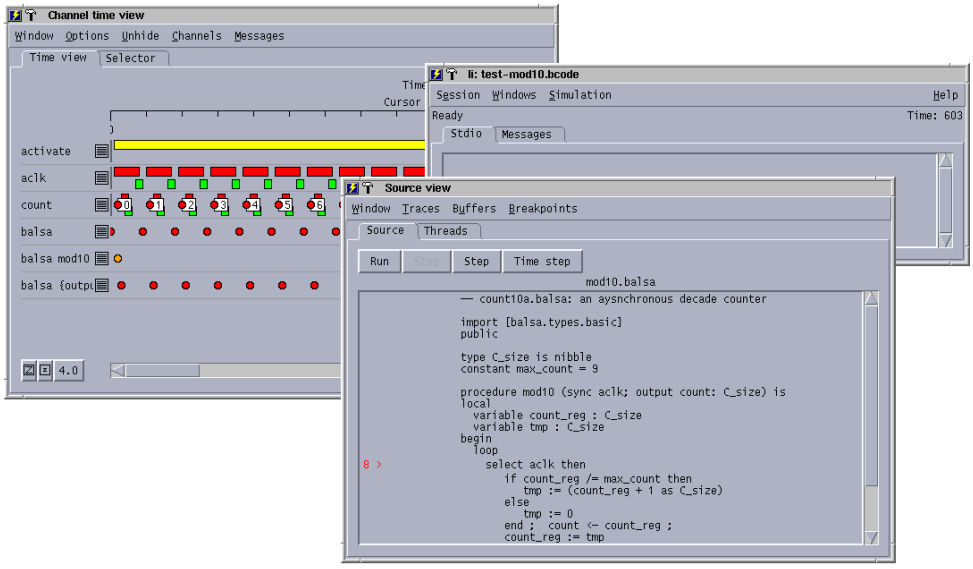


**Figure 7. LARD simulation of modulo-10 counter**

count ranges, with the remaining channels having their address spaces limited to the portion of the memory map occupied by the synchronous peripherals in order to reduce the amount of state needed to be stored in the controller.

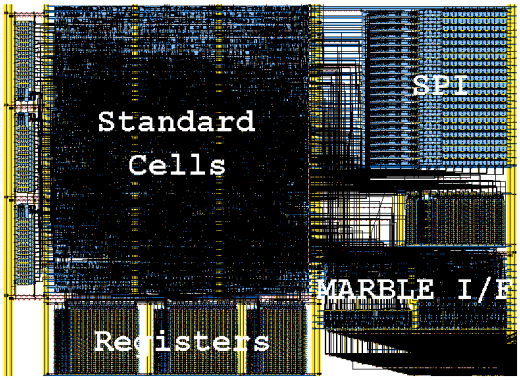


**Figure 8. AMULET3i DMA controller layout**

The DMA controller has been implemented in 0.35 µm Si CMOS, as shown in figure 8, using a mixture of full-custom layout (the register blocks and Synchronous Peripheral Interface (SPI) multiplexer cells) and standard cells provided by ARM Ltd. and based on their process independent process rules. Additional cells were designed in-house for distinctively asynchronous elements such as mutual exclusion elements and Muller-C elements. The completed controller occupies just over 2.1 $mm^2$ of silicon area consisting of 70 000 transistors (1 $mm^2$ of which is standard cell synthesised Balsa), contains nearly 2000 bits of register state and can perform a complete read; write transfer pair in around 90 ns which comfortably outperforms the synchronous bridge to which it transfers most of its data. Design was undertaken using Compass Design Automation tools with the verifying simulations being performed on capacitance-extracted layout using the TimeMill simulator.

Handshake channels were implemented using the 'bundled data' methodology in which data is accompanied by separate request and acknowledge control signals with timing constraints on the relationships between data changes and control signalling. These relationships need to be verified on the implementation and so timing verification was performed by automated examination of simulation results. A conservative design approach also reduced the likelyhood of timing violations considerably. It is hoped that future Balsa back-ends will have integrated timing analysis and can so produce more 'risky' circuits which can be post-layout verified to produce smaller and faster implementations.

During the design process the controller was re-implemented a number of times to improve performance, reduce area and to change behaviour to suit the customer. These re-implementations typically took less than a few hours to implement (much of which was spend reorganising the non-synthesised portions of the design) so vindicating the use of synthesis.

## 4. Conclusions

Balsa has already proven itself useful in rapidly implementing designs for moderate speed peripherals. Future work on Balsa will include improvements in the richness of the handshake component set used for compilation, work on data path synthesis and DI codes in data paths, improvements in the simulation environment and the addition of timing analysis to the back-end tools.

### References


[1] AMULET2 Halt Instruction. UK Patent No. 9620973.9.

[2] F. Bennett, D. Clarke, J. Evans, A. Hopper, A. Jones and D. Leask. Piconet – Embedded Mobile Networking. *IEEE Personal Communications* **4(5)**, Pages 8–15 (October 1997).

[3] Kees van Berkel, M. Rem. VLSI Programming of Asynchronous Circuits for Low Power. In G. Birtwistle, A. Davis (Eds), *Asynchronous Digital Circuit Design*. Workshops in Computing – Proceedings of Asynchronous Design Workshop Banff, Alberta 28 Aug. - 2 Sep. 1993. Springer, 1993.

[4] W. J. Bainbridge, S. B. Furber. Asynchronous Macrocell Interconnect Using MARBLE. In *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'98*. Department of Computer Science, The University of Manchester, 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] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. In *IEICE Transactions on Informations and Systems*, pages 315–325, 1997.

[7] EDIF Library of Parameterized Modules. URL *http://www.edif.org/lpmweb/*.

[8] R. Fuher, S. Nowick et. al.. MINIMALIST: An Environment for the Synthesis, Verification and Testability of Burst Mode Asynchronous Machines. Tech. Rep. (1999), Department of Computer Science, Columbia University, USA.

[9] S. B. Furber, J. D. Garside, S. Temple, J. Liu, P. Day, N. C. Paver. AMULET2e: An Asynchronous Embedded Controller. In *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'97*. Department of Computer Science, The University of Manchester. Cogency Technology Inc, April 1997.

[10] J. D. Garside, S. B. Furber, S-H Chung. AMULET3 Revealed. In *Fifth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'99*. Department of Computer Science, The University of Manchester, March 1999.

[11] 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, The Netherlands, March 1998.

[12] J. Kessels. VLSI Programming of a Low-Power Asynchronous Reed-Solomon Decoder. In *Asynchronous Design Methodologies*, pages Pages 44–52, May 1995.

[13] J. Kessels. Designing Asynchronous Standby Circuits for a Low-Power Pager. In *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'97*. Philips Research Laboratories, The Netherlands, April 1997.

[14] D. J. Kinniment, B, Gao, A. V. Yakovlev, F. Xia. Towards Asynchronous A-D Conversion. In *Fourth International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC'98*, pages Pages 206–215, March 1998.

[15] A. M. G. Peeters. Tangram99 talk. In *ACiD WG Workshop – University of Newcastle upon Tyne, UK*. Edited by: M. B. Josephs and A. V. Yakovlev. Philips Research, 18-19 January 1999.

[16] Sharp Digital Information Products – Data-Driven Media Processor. URL *http://www.sharpsdi.com/DDMPhtmlpages/DDMPmain.html*.

[17] Theseus Logic Inc. URL *http://www.theseus.com*.