CHAIN: A DELAY-INSENSITIVE CHIP AREA INTERCONNECT

THE INCREASING COMPLEXITY OF SYSTEM-ON-A-CHIP DESIGNS EXPOSES THE LIMITS IMPOSED BY THE STANDARD SYNCHRONOUS BUS. THE AUTHORS PROPOSE A MIXED SYSTEM AS A SOLUTION.

John Bainbridge
Steve Furber
University of Manchester

A globally shared bus increasingly cannot meet the demands of system-on-a-chip (SOC) interconnects because the high wire loads and resistance levels result in slow signal propagation. A popular alternative, using unidirectional point-to-point connections, and multiplexers, requires even more chip space and still suffers from many of the same problems (for example, difficulties in timing validation and connecting devices running from unrelated clocks). Furthermore, to continue increasing the number and variety of macrocells embedded on a single chip will require interconnects with greater flexibility than today's synchronous, core-specific system buses.

The adoption of on-chip networks as the solution to growing SOC interconnect demands, presented by both current and projected integration levels,[1] raises the question of which clocking strategy to use. A higher clock frequency produces better performance, but by definition, the global interconnect spans the entire chip—exactly the situation that leads to clock skew problems. Clock management of such a network is problematic at best.

Using self-timed techniques for the network on a chip eliminates these problems. This leaves only the issue of interfacing synchronous and self-timed circuits, which is a well understood discipline for which standard solutions exist.[2]

One-hot encoding

Our network on a chip, Chain, uses a delay-insensitive data encoding combined with a return-to-zero signaling protocol, on links with one acknowledge wire and five forward-going wires. Table 1 shows this one-of-five data encoding where only one wire in a group is allowed to signal data at any one time (known as one-hot encoding). During normal data transmission, one of the wires (d3, d2, d1 or d0) transmits two bits per cycle. The end-of-packet (EOP) signal separates consecutive data blocks. This encoding requires only a simple five-input OR gate to detect valid data at the receiver. Multiple parallel links can satisfy higher bandwidth requirements.

Design advantages

Rather than sharing an acknowledge wire across many bits of forward-going data, our approach runs a dedicated acknowledge wire for each one-of-five signaling group (giving six wires for every link). This additional wiring avoids the need for trees of gates in the critical path to merge multiple acknowledges into a single wire.

Interconnect delays and crosstalk are becoming worse with every CMOS process shrink, and repeaters have to be inserted at intervals along long wires to repower the links.[3] Using pipeline latches instead of buffers allows bandwidth improvement of such links with little impact on their latency.

Buffer and pipeline techniques apply equally to clocked and asynchronous circuits. With or without clocks, performance is limited by

0272-1732/02/$17.00 © 2002 IEEE

Table 1. One-of-five data encoding.

End-of-packet signal	d3 wire	d2 wire	d1 wire	d0 wire	Information transferred
1	0	0	0	0	End of packet
0	1	0	0	0	Two-bit data value 11
0	0	1	0	0	Two-bit data value 10
0	0	0	1	0	Two-bit data value 01
0	0	0	0	1	Two-bit data value 00
0	0	0	0	0	Idle state

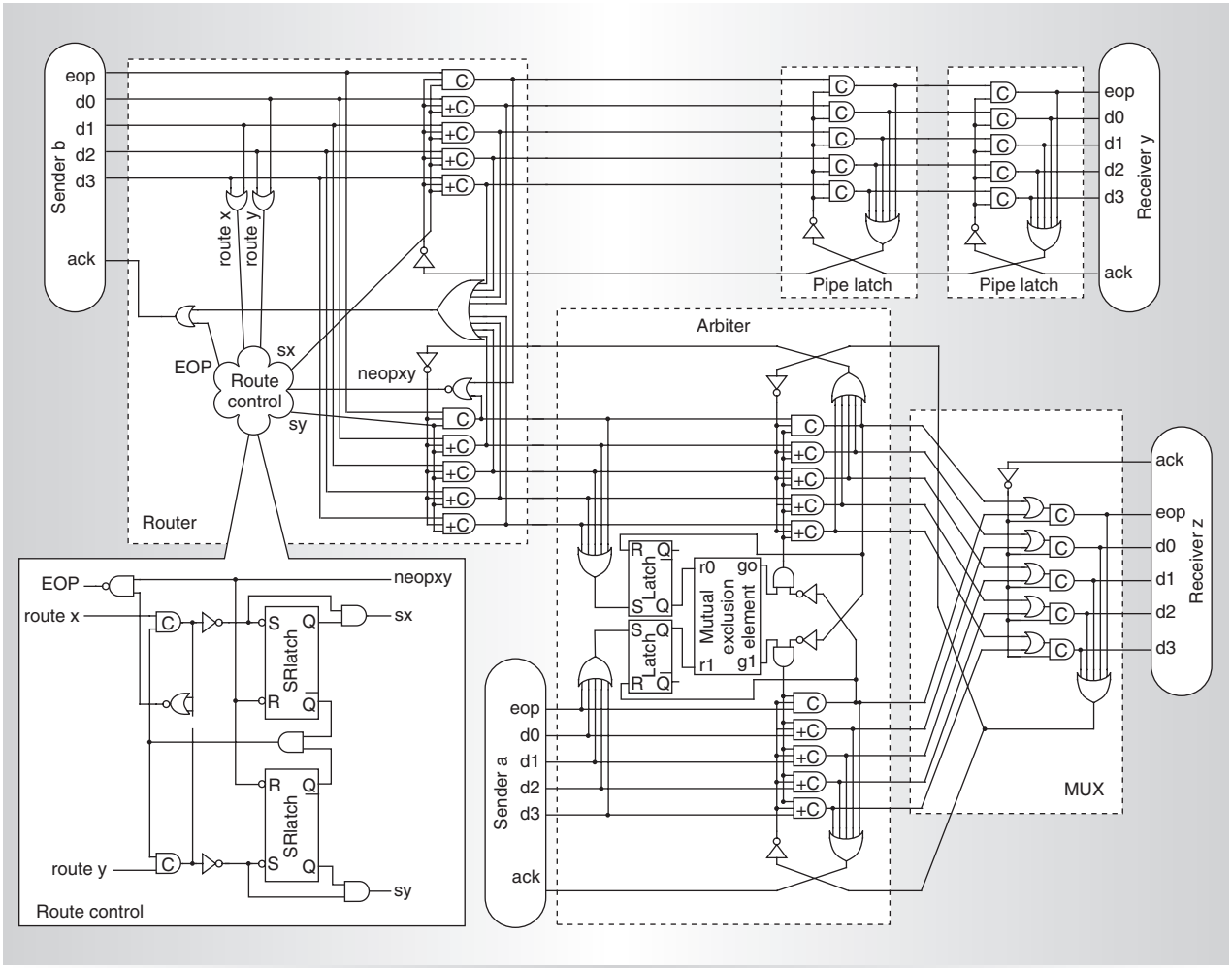

Figure 1. Collage of Chain network components.

the slowest stage. The key difference is that with a synchronous approach, the entire interconnect must be operated from the same clock, or a multiple thereof, whereas the asynchronous approach is self-regulating, operating as fast as the paths allow. The "Self-timed circuits" sidebar (next page) gives a detailed explanation of how asynchronous circuits work.

The pipe latches in Figure 1 represent the self-timed latch stages for the one-hot links.[4] The loop between these latches can be thought of as a ring oscillator, interlocked with the preceding and following stages via the Muller C-elements, with the minimum oscillation period determined by the two C-elements, the OR gate, the inverter, and the

Self-timed circuits

Self-timed (asynchronous) circuits exchange information using a handshake to explicitly indicate the validity and acceptance of data. This is in contrast to the synchronous design style, which uses a globally distributed clock signal to indicate moments of stability of the data.

There are a number of alternative asynchronous design styles differing in how they indicate data validity.[1-3] The fabrication of a range of large-scale designs proves the viability of these approaches.[4,5] Some styles base the handshake upon a matched-delay path representing the slowest part of the data path, but these suffer from the same timing validation problems as synchronous design. We use, in Chain, the delay-insensitive style in which the validity is transmitted implicitly in the data encoding. This avoids much of the need for timing analysis, giving designs that operate correctly regardless of the delay in the interconnecting wires, although coming with an increased resource cost. Many data encoding schemes can be used in this manner,[3] including the simple dual-rail approach, which uses two signal wires per data bit; for our example we will call the wires d0 and d1. The idle state is when both signals are low and convey no information. If d0 is asserted then a logic 0 is transmitted; if d1 is asserted then a logic 1 is transmitted. At any time, at most one of the signals can be asserted, conveying both the data value and, by its assertion, indicating data stability. The receiver uses another wire, the acknowledge, to signal receipt of the data, and then in a return-to-zero phase the data and the acknowledge wires return to the idle state.

Usually, many pairs of data wires share a single acknowledge. The channel cost of this approach for an n-bit data path is $2n + 1$ wires, n2-input **OR** gates and an n-input Muller C-element tree. This is the price paid to avoid post-layout timing analysis and validation. A Muller C-element is like a stateful AND gate. Its output is asserted when both inputs are high; and deasserted when both inputs are low. With differing inputs, the gate output retains its previous value. Although this requires state in the gate, they are still fair-

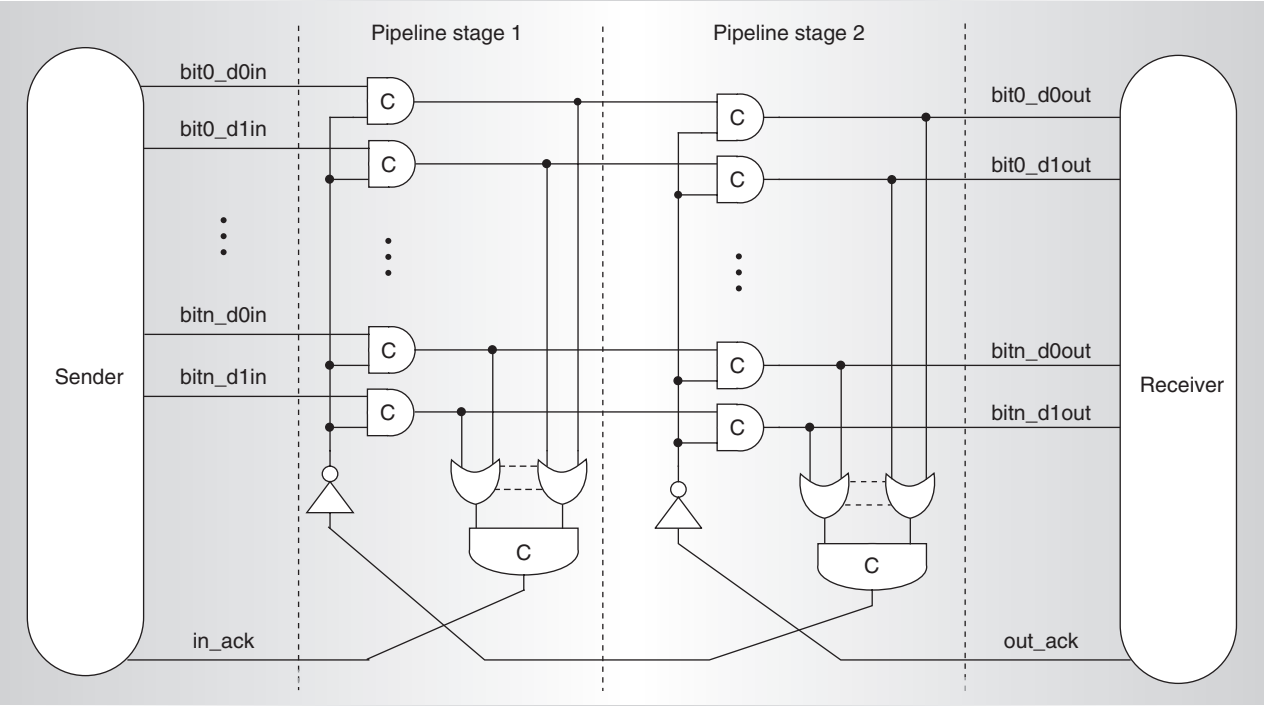

Figure A. Dual-rail, self-timed pipeline channel.

lengths of the wire between the latch stages. The other modules shown in Figure 1 are derivatives of this simple latch stage introduced to perform switching functions.

Crosstalk

Using a return-to-zero, one-hot code has the further advantage of minimizing crosstalk. The increasing importance of this issue is apparent when you consider that the variation in signal delay (for 0.35-micron technology) on a 10-millimeter wire is between 0.5 and 2.5 nanoseconds (ns) depending on the behavior of its environment.[5] First, only one wire in a link can transition at a given time, meaning that signals within a link will never transition in opposing directions at the same time. Second, detrimental interaction between physically adjacent one-of-five links will occur infrequently because of sparse coding and asynchronicity, which

ly inexpensive, with a two-input C-element requiring only eight transistors.

Figure A shows a sender connected to a receiver through two pipeline latch stages using the dual-rail signaling scheme. Each latch stage has two key parts: the C-elements providing storage and synchronization of the data path with the acknowledge flow-control signal from the following stage; and the completion detection tree, composed of OR gates feeding into a C-element to generate the acknowledge signal for use by the previous stage. The C-elements behave like AND gates except that, once asserted, their output remains so until ALL of the inputs are low (not just one of them). This behavior means that we can think of circuits such as that shown in Figure A as a series of interlocked ring oscillators, as is apparent if one follows the signal flow from the upper Stage 1 C-element output through the uppermost Stage 2 C-element to the Stage 2 completion detector and the Stage 1 inverter back to the other input of the upper Stage 1 C-element – a loop with a total of nine logical inversions. As such, the circuits are self-regulating, their speed determined by the slowest stage without the use of a global clock to regulate every stage.

References

1. A.M.G. Peeters, *Single-Rail Handshake Circuits*, doctoral dissertation, Technische Universiteit Eindhoven, Eindhoven, Netherlands, 1996.
2. I.E. Sutherland, ``Micropipelines,'' *Comm. ACM*, vol. 5, no. 6, June 1989, pp 720-738.
3. T. Verhoeff, "Delay-Insensitive Codes—An Overview," *Distributed Computing*, vol. 3, no. 1, 1988, pp. 1-8.
4. Garside et al., "AMULET3i an Asynchronous System-on-Chip," *Proc. 6th IEEE Int'l Symp. Asynchronous Circuits and Systems* (Async), IEEE CS Press, Los Alamitos, Calif., 2000, pp. 162-175.
5. M. Renaudin, P. Vivet, and F. Robin, "ASPRO-216: A Standard Cell QDI 16-Bit RISC Asynchronous Microprocessor," *Proc. 4th IEEE Int'l Symp. Asynchronous Circuits and Systems* (Async), IEEE CS Press, Los Alamitos, Calif., 1998, pp. 22-31.

makes it unlikely that adjacent links will transition at exactly the same time.

Link speed

On 0.35-micron technology, simulations show a throughput of around 700 megabits per second (Mbps) per link, with more than 1 Gbps per link projected for 0.18-micron CMOS technology—using suitable link lengths to minimize end-to-end latency. This corresponds to 120 Mbps per wire on 0.35-micron CMOS technology and 160 Mbps per wire on 0.18-micron CMOS technology.

Switched Chain networks

Switched networks require additional circuitry to handle steering, multiplexing, arbitration, and route setup and teardown. We can easily adapt the basic one-of-five pipeline latch to support these functions, and distribute them throughout the interconnection system illustrated in Figure 1.

To pass data from a sender to a receiver through a network of links connected through pipeline, multiplexing, and steering latches, a route has to be determined. Figure 1 shows how a modified latch allows the multiplexing of one of two inputs onto an output. If the environment cannot guarantee mutual exclusivity of activity on the incoming links, a link arbiter (also shown in Figure 1) is required immediately prior to the multiplexer. This arbiter's operation is entirely self-timed, using a mutual exclusion element[6] to resolve metastability. Figure 1 shows the corresponding implementation of the steering unit for directing a symbol to one of two alternative outputs.

Route setup

In our Chain implementation, each sender has full knowledge of the network topology and can thus determine the routing. This information is then encoded in a series of routing symbols at the start of the transmitted packet.

Figure 1 shows a suitable circuit fragment for tapping-off the first incoming symbol to the steering unit and using it to direct the subsequent symbols, resetting on detection of the EOP symbol. This circuit fragment imposes a small additional delay on route setup but does not impede performance for subsequent data symbols; this is also the case for the arbiter module. To support packet-based operation the arbiter ensures that packets remain intact (not fragmented). Hence, arbitration can only occur following reset or passage of an EOP symbol.

Route teardown

After a group of symbols forming a packet passes through a network node, that node can be released to service a new incoming packet. This occurs when the node observes an EOP symbol, transmitted on the EOP wire. In the early stages of Chain's development, we experimented with using only four forward-going data wires, using fixed-length packets, and thus

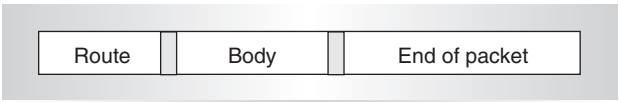

Figure 2. Chain's packet format.

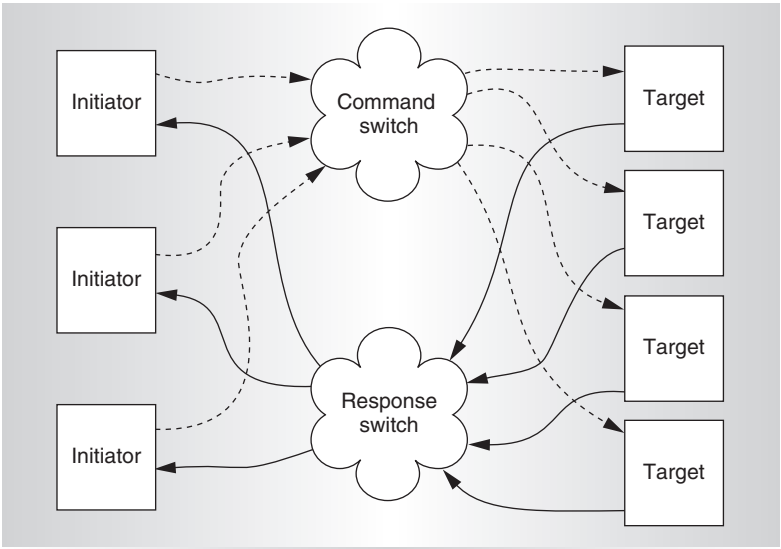

Figure 3. Chain's on-chip network connectivity.

eliminating the need for the EOP wire. That scheme, however, resulted in significantly more complex switching circuits that had to count the number of symbols they passed.

Packet length

Our current Chain implementation accommodates variable length packets through the explicit EOP signaling. For route setup we distribute as much of the control as possible, so every packet must carry routing information at its head. Figure 2 shows this packet format.

Parameterizable performance

Although individual Chain links deliver significant throughput, many systems require greater than the 700 Mbps available over a single link (on 0.35-micron CMOS technology). Bandwidth is improved in the Chain architecture by ganging links together. The packet is mapped onto a gang of links with the ROUTE and EOP fields transmitted down every link, and the BODY field divided across the links.

A 32-bit packet body transmitted over 16 links in parallel incurs only the route setup time and the end-to-end latency. At the other extreme, transmitting the same packet over a single link as 16 sequential symbols incurs an additional 48 ns of latency (around 3 ns per symbol on 0.35-micron technology).

The multiplexing and demultiplexing required to transmit a packet body serially is derived from the multiplexer and steering circuits, shown in Figure 1, by adding a few gates in the acknowledge paths to alternate between the two available ports. This multiplexing simplicity, combined with self-timed links, which always run as fast as the source and destination allow, means that we can seamlessly tailor the bandwidth of different parts of the network to suit the units connected to them.

System-on-a-chip interconnect

Designers can use a switched network architecture to create networks of any topology, including rings, meshes, and multiplexer/demultiplexer systems that approximate the behavior of a shared bus.

Current SOC designs require the connection of macrocells such as processors, direct memory access controllers, test interfaces, and a hoard of peripherals (memories, universal asynchronous receiver/transmitters or UARTs, and so on). The interconnect must pass commands from the initiator to the target device, and then return a response in the opposite direction. The open core protocol neatly specifies this, albeit using a clock signal.[8] A processor reads data from a memory by transmitting an address and the memory responds with the read data. For performance reasons, most on-chip bus implementations transmit the command and response over separate pathways.[9,10] With Chain, as Figure 3 illustrates, this corresponds to two separate switched networks that are minimally coupled at the initiator and target through the use of split transactions for all operations—inexpensive and easy to achieve in an asynchronous design.[10]

Message format

Higher-level protocols can be layered upon the basic connectivity provided by the switching of variable-length packets we have described. Several message formats are thus defined, all comprising a 16-bit header followed by payload information (rounded to a multiple of 16 bits), as shown in Figure 4. Some of the shorter messages have no payload, just a message header.

Currently we use five different types of messages with Chain:

- Command—sends the address and write data (if any) to the target.
- Normal response—returns success confirmation and read data (if any) to the transaction initiator.
- Notify error—indicates trouble at the target.
- Notify accept—indicates transaction accepted by a slow target.
- Notify defer—indicates that the target rejected the transaction.

In normal operation, the initiator issues a command (including an address and optionally, write data) that a target accepts for processing. Upon completion of the processing, the target replies with a normal response to indicate success and to return any requested read data.

In the (hopefully rare) situations when the target encounters problems in processing the command, the notify-error message aborts the operation precisely, which tells the initiator exactly which command caused an exception.

For slow targets, the notify-accept message lets the target inform the initiator that the command will require extra time to process.

The notify-defer message has a very specific use (and in many designs might not be required). It lets the target reject a command, thus signaling a temporary failure. This behavior is useful when a network deadlock scenario arises; such as when using a multiplex/demultiplex network topology connected through a bridge to another multimaster system interconnect—such as the AMBA bus from ARM[9]—or another Chain network. However, this type of hardware retry mechanism results in inefficient power and network bandwidth use and should be avoided whenever possible.

Multimessage packets

Atomic sequences of messages can be passed across a Chain fabric using a multimessage packet. Each message has a header and possibly a payload (depending on the message). The full packet format, where [ ] represents options and * represents repetition, is then:

ROUTE HEADER [PAYLOAD [HEADER]] * EOP.

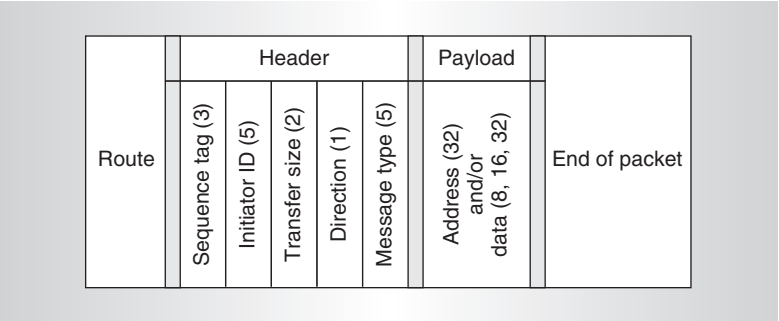

Figure 4. Message format (number of bits shown in parentheses).

Multimessage packets are required to support semaphore operations. They also reduce the impact of the route setup stage on overall throughput, although this can increase latency for other waiting senders.

Prototype implementation

Our first implementation of Chain is in a smart card chip (in fabrication at the time of publication).[11] This low-performance system uses a fairly simple Chain network with:

- two ganged links (12 wires per pipeline stage) for each of the forward (command) and reverse (response),
- a multiplexer/demultiplexer topology, and
- one outstanding command per initiator.

In this chip, all of the clients connected to Chain are asynchronous macrocells, including the two synthesized processor cores and the external UART—although the UART uses a clock for its external connection. Figure 5 (next page) shows the full network configuration for the command path. The response path is a mirror image of this, with steering blocks replaced by arbiter/multiplexer combinations and vice versa.

The system shown in Figure 5 has three initiators: the two ARM instruction-set-compatible microprocessors, and an external test interface allowing fabrication test access to the embedded macrocells via the network. Testing the processors consists of loading code into the RAM and then running it.

The *bare off-chip interface* in the system allows direct connection of the processors to an off-chip memory subsystem for test purposes. This interface only uses one Chain link

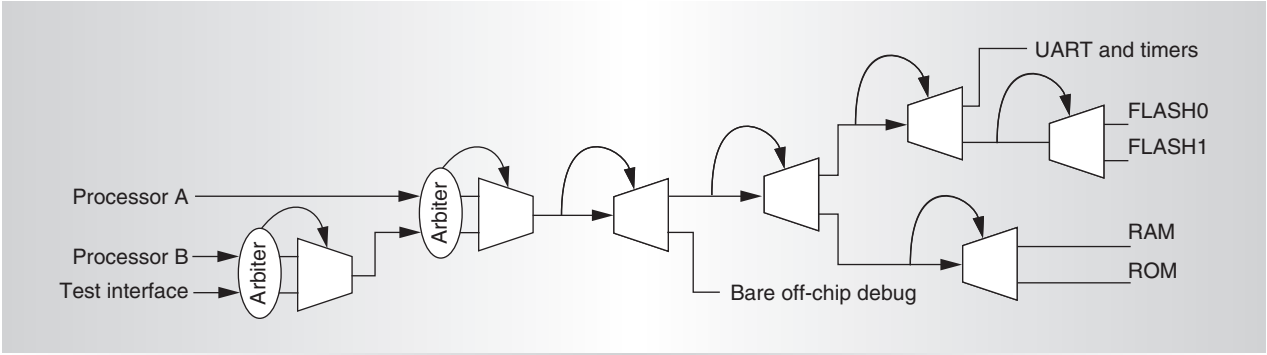

Figure 5. Chain network command path in an experimental smart card chip.

Table 2. Chain component sizes.

Component	No. of transistors
Steering	250
Arbiter	250
Multiplexer	100
Pipeline-latch repeater	70
Initiator interface	10,250
Target interface	11,000
Initiator address decoder/route encoder	1,250
Command and response (two-link wide) fabric	9,500

in each direction, whereas the rest of the network uses two ganged links throughout. For normal on-chip operation, this link is unused and has no impact on performance.

Network costs

The network fabric components are relatively simple, around 20 gates each. Table 2 shows a summary of these costs (as a transistor count) and those of the initiator and target interfaces.

The whole Chain logic for the smart card system implementation, including the address decoders, the fabric, three initiator interfaces, and seven target interfaces uses 115,000 transistors. These are scattered throughout the layout, each interface place-and-routed with its associated device.

Performance

The worst-case performance of the smart card's network configuration occurs when the processor core is accessing one of the flash memory blocks, requiring the command and response packets to pass through the most hops and travel the greatest distance. At the 1-Gbps per link rate of the 0.18-micron CMOS implementation, command route setup incurs 4 ns of latency in addition to the transit time for the signal to propagate through the logic and wires. A read-command message containing a 16-bit header and 32 bits of address requires an additional 24 ns, whereas a write command with an additional 32 bits of write-data would require an additional 40 ns instead.

The latency imposed by the network on returned read data, in addition to the basic propagation delay, is due to three arbitrations, one route-setup symbol, and the passage of the 16-bit message header and the 32 bits of read data. This gives a total time of 26 ns.

With only one outstanding command issued at a time by the processor, the maximum memory access rate to the flash memory blocks is about 20 million transfers per second. This limits the processor to fewer than 20 million instructions per second (once the flash access time is taken into account), which is more than sufficient for current smart card applications.

This smart card system uses only two ganged links for each of the forward and return paths, giving modest performance at low cost. With wider gangs of links, our projections show that Chain gives better throughput compared to current synchronous interconnect operating at frequencies over 100 MHz,[9] and, of course, the latency also improves with wider gangs. For example, a 48-wire gang of eight links would give 1-Gbytes/s throughput. Our approach achieves this while avoiding clock skew problems and eliminating the need for post-layout timing validation.

Although the first implementation of Chain is in an entirely asynchronous system, techniques for interfacing synchronous and asynchronous systems are well known.[2] Wrappers, based on synchronization or pausible clocking techniques, can connect synchronous IP blocks to a self-timed interconnect such as Chain. It is now possible to construct large-scale SOCs without encountering the problems of synchronous interconnects. Of course, this also opens the door for using a mixture of synchronous and self-timed macrocells on the same SOC. MICRO

Acknowledgments

We gratefully acknowledge financial support from the United Kingdom's Engineering and Physical Sciences Research Council (EPSRC) grant GR/R47363/ 01, and from Theseus Logic Inc.

References

1. W.J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. Design and Automation Conf.* (DAC), ACM Press, New York, 2001, pp. 684-689.
2. S. Moore et al., "Point to Point GALS Interconnects," *Proc. 8th IEEE Int'l Symp. Asynchronous Circuits and Systems* (Async), IEEE CS Press, Los Alamitos, Calif., 2002, pp. 69-75.
3. H.B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, Boston, 1990.
4. M. Renaudin, P. Vivet, and F. Robin, "ASPRO-216: A Standard Cell QDI 16-Bit RISC Asynchronous Microprocessor," *Proc. 4th IEEE Int'l Symp. Asynchronous Circuits and Systems* (Async), IEEE CS Press, Los Alamitos, Calif., 1998 pp 22-31.
5. P. Nordholtz et al., "Signal Integrity Problems in Deep Submicron Arising from Interconnects between Cores," *Proc. 16th IEEE VLSI Test Symp.* (VTS), IEEE CS Press, Los Alamitos, Calif., 1998, pp. 28-33.
6. C. Seitz, "System Timing," *VLSI Systems*, C. Mead and L. Conway, Addison-Wesley, Boston, 1980.
7. W.J. Bainbridge and S.B. Furber, "Delay Insensitive System-on-Chip Interconnect using 1-of-4 Data Encoding," *Proc. 7th IEEE Int'l Symp. Asynchronous Circuits and Systems* (Async), IEEE CS Press, Los Alamitos, Calif., 2001, pp. 118-126.
8. *Open Core Protocol Specification,* OCP International Partnership (OCP-IP), Portland, Ore.; http://www.ocpip.org (current Aug. 2002).
9. *Advanced Microcontroller Bus Architecture (AMBA) Specification Rev. 2.0*, ARM Ltd., document no. IHI-0011A, May 1999; http://www.arm.com (current Aug. 2002).
10. J. Bainbridge, *Asynchronous System-on-Chip Interconnect*, Springer, Heidelberg, Germany, 2002.
11. L.A Plana et al., "SPA—A Synthesizable Amulet Core for Smartcard Applications," *Proc. 8th IEEE Int'l Symp. Asynchronous Circuits and Systems (*Async), IEEE CS Press, Los Alamitos, Calif., 2002, pp. 201-210.

John Bainbridge is a research associate in the Department of Computer Science at the University of Manchester, England. His interests include asynchronous logic and SOC design. Bainbridge has an M.Eng in electronic systems engineering and a PhD in computer science from the University of Manchester. His doctoral thesis was the winner of the 2001 Distinguished Dissertation Competition awarded jointly by the British Computer Society and the Conference of Professors and Heads of Computing. He is a member of the IEEE and the IEE.

Steve Furber is the ICL Professor of Computer Engineering in the Department of Computer Science at the University of Manchester. His research interests include asynchronous logic design and power-efficient computing. Furber has a BA in mathematics and a PhD in aerodynamics, both from the University of Cambridge, England. He is a fellow of the Royal Society, the Royal Academy of Engineering, and the British Computer Society; a chartered engineer; and a senior member of the IEEE.

Direct questions and comments about this article to John Bainbridge, Univ. of Manchester, Oxford Rd., Manchester, M13 9PL, UK; jbainbridge@cs.man.ac.uk.