Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 200 (2008) 17–32
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Performance Evaluation of Elastic GALS Interfaces and Network Fabric

Junbok You Yang Xu Hosuk Han Kenneth S. Stevens

Electrical and Computer Engineering
University of Utah
Salt Lake City, U.S.A
junbok.you@utah.edu yang.xu@utah.edu hosuk.han@utah.edu kstevens@ece.utah.edu

Abstract

This paper reports on the design of a test chip built to test a) a new latency insensitive network fabric protocol and circuits, b) a new synchronizer design, and c) how efficiently one can synchronize into a clocked domain when elastic interfaces are utilized. Simulations show that the latency insensitive network allows excellent characterization of network performance in terms of the cost of routing, amount of blocking due to congestion, and message buffering. The network routers show that peak performance near 100% link utilization is achieved under congestion and combining. This enables accurate high-level modeling of the behavior of the network fabric so that optimized network design, including placement and routing, can occur through high-level network synthesis tools. The chip also shows that when elastic interfaces are used at the boundary of clock synchronization points then efficient domain crossings can occur. Buffering at the synchronization points are required to allow for variability in clocking frequencies and correct data transmission. The asynchronous buffering and synchronization scheme is shown to perform over four times faster than the clocked interface.

Keywords: System on chip, network on chip, synchronizer, interface architecture, asynchronous FIFO

1 Introduction

Power and time to market requirements dictate the need to operate many of the independent blocks on our chip at a frequency and power optimal for the particular design and work load. This results in designs that contain many unrelated frequency islands [10]. Unfortunately, designing a communication fabric that interfaces and synchronizes to these frequency islands has been challenging and costly in terms of power and delay [5]. The cost of communication across an elastic network fabric with various interface designs is evaluated. We report on simulations carried out on the layout of a test chip fabricated in the AMIS 500nm three metal process technology. The simulations will be validated against the silicon when it returns

[1] This work was supported in part by Semiconductor Research Corporation under Contract No. 2006-TJ-1424.

1571-0661 © 2008 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.entcs.2008.02.004

from the foundry. The design measures two main components: the efficiency of a new elastic protocol applied to a network fabric, and the overheads of synchronizing into a clocked domain using a GALS protocol. The new network fabric shows near 100% link utilization can be achieved, and that real time constraints can be achieved even under full network saturation. A comparison between three synchronization interfaces shows that asynchronous interface logic can be nearly as efficient as a fully synchronous system, and that clocked synchronization protocols can show substantial overhead.

1.1 Latency Insensitive Protocols

The potential design space of a Network-on-Chip (NoC) is very large, and includes topology choice (mesh, torus, star, etc.), circuit switched or packet switched, and other parameters (link widths, frequency, etc.). The design space is further enhanced when we add a globally asynchronous and locally synchronous (GALS) methodology [11,1]. Given this design diversity, perhaps one the most important design choice is the protocol for the communication fabric and the interface protocol between the network and the intellectual property (IP) cores. Design performance, complexity, and the application space are determined by the channel protocol employed. A new class of protocols has recently emerged from a wealth of GALS designs called *latency insensitive protocols* (LIP), pioneered by Carloni [2,3]. There are various classes of such protocols but all use a handshaking protocol similar to traditional asynchronous design to implement flow control and data buffering. LIP protocols are traditionally applied to clocked systems.

Latency insensitive protocols have some potential advantages in GALS systems over other designs due to two main factors. First, since these protocols have been applied to clocked design, there is the potential of implementing efficient interfaces to the clocked IP that doesn't require stoppable clocks and allows the IP cores to stall under network congestion. This could be particularly advantageous if both the IP cores and the network fabric use identical or compatible protocols. Second, the efficient buffering of LIP protocols under stall conditions has a potential of creating a very low overhead, high throughput dynamic network fabric. We test this hypothesis by implementing the network fabric, synchronizing interfaces, and IP interface with a latency insensitive, or *elastic* protocol.

Elastic systems are just like clocked systems in that they consist of a collection of modules and channels. However, the elastic communication channels have two control wires, *valid* and *stall* that implement a handshake between the sender and receiver. The data and valid signals propagate down the pipeline, and stall propagates backwards up the pipeline. We have chosen a protocol similar to the Synchronous Elastic Flow (SELF) channel protocol [4].

The synchronization between source and destination channels in SELF are very similar to the Synchronous Interlocked Pipelines [8]. Interlocked pipelines implement independent control for each of the two latches that comprise a flip-flop. When a stall occurs during data transfer, the independent latch control permits the second data arriving at the flop to be stored in the second latch, allowing a flop to store two

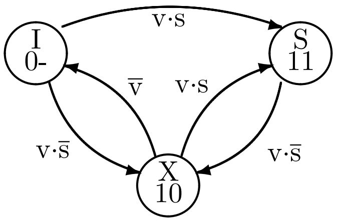

Fig. 1. Elastic Channel Protocol, state: {v s}

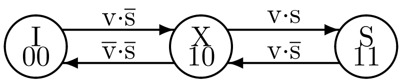

Fig. 2. Phased Elastic Channel Protocol, state: {v s}

data items. The stall signal is concurrently propagated up the pipeline. The stall now arrives at the previous stage just in time to ensure that a second data arriving in that flop will be stored in the second latch, and so forth. These protocols allow stalling a pipeline to occur with zero overhead to the clock frequency in an elastic clocked system.

The SELF channel protocol is shown in Figure 1. We have completed the initial development of a complementary new *phase* channel protocol (pSELF) shown in Figure 2. Both protocols are idle in the 'I' state, actively transfer data each cycle in the 'X' state, are stalled in the 'S' state. Each of these states are labeled with the values of the valid (v) and stall (s) signals. In the SELF protocol both valid and stall signals change on the same edge of the clock, in the phase protocol they change mutually exclusively on different clock phases. This avoids the redundant idle state that is both stalled and not valid.

Both of these protocols allow the two latches in a clocked design to be split and separated by long communication lines with the associated wire delay. The SELF protocol has the advantage of permitting a nearly full cycle of latency between any two latches, whereas the pSELF protocol only tolerates one phase of latency. However, the pSELF protocol has the advantage that it can directly communicate with an asynchronous channel where the request and acknowledge signals switch in alternating phases. The pSELF elastic protocol now supports direct communication with clocked or asynchronous logic, and can be directly mapped and synthesized into either a clocked or asynchronous network fabrics [6]. This will allow us to make good apples-to-apples comparisons between GALS systems using a clocked or asynchronous network fabric (or a clocked network and asynchronous IP).

1.2 Synchronizing Interfaces

The second main focus of the chip is the synchronizing interface into clocked domains. These synchronizing interfaces are very challenging to design and have significant performance implications [7].

Our GALS network interface is based on the fundamental assumption that the synchronous islands contain an elastic channel interface protocol. Given such a protocol the main cost of interfacing to an asynchronously operating network environment is the cost of synchronization and buffering.

To this end we have implemented three simple interfaces between the network and our clocked processing elements: 1) a fully synchronous interface where the network and processing elements must all operate on the same low skew clock, 2) a clocked synchronizer and synchronous handshake protocol with synchronous buffering, and 3) a novel fast synchronizer with asynchronous buffering. We have built all of the buffers as simple linear flow-through FIFOs, where the interfaces contain four data words of buffering. The fully synchronous interface has no need for buffers so they are not included.

2 Network Fabric

The network fabric in this work uses design targets that differ significantly from other Network on Chip (NoC) designs, including: a) A non-redundant network topology. b) No multi-word packets - each transmission is a single data word containing all necessary routing information. c) Simple high throughput network routers and buffers. d) Both clocked and fully asynchronous realizations of the NoC fabric. This produces a fabric that has ultra low latency, high throughput, and a static worst case latency for all transmissions (assuming sufficient buffering exists in the network interfaces).

2.1 Network Components

All of the links in our network are bidirectional. Bidirectional network links consist of two independent elastic channels that can concurrently transmit data in opposite directions. The network fabric is implemented using two components: a phase elastic half buffer (pEHB) and a binary routing buffer, or ⊤ element. The pEHB, shown in Figure 3, contains one of the two latches in a flip-flop pipeline stage with its associated logic implementing the phase based SELF control protocol.

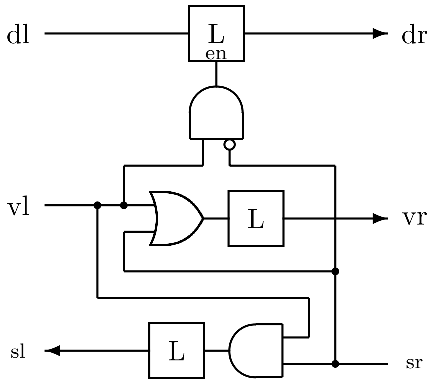

Fig. 3. Phase Elastic Half Buffer Logic

The router elements behave like binary switch sending an input token out one of two output paths. Their configuration looks like a ⊤. Complicated network topologies are composed out of these simple binary routing switches. The router consists of three switch blocks and three data merge components shown in Figure 4. The switch blocks steer data on incoming channels to one of the other two outgoing channels. The merge elements allow each outgoing channel to be shared by the two incoming arms in the ⊤. Note that two requests can arrive at a join module in the same clock cycle. Therefore the join module requires arbitration and buffering. Fair arbitration protocols are used between the incoming channels for both the clocked and asynchronous designs. The merge blocks contain an elastic half buffer. The pEHBs provide buffering on the three outgoing channels. This results in a ⊤ router where all ports are bidirectional.

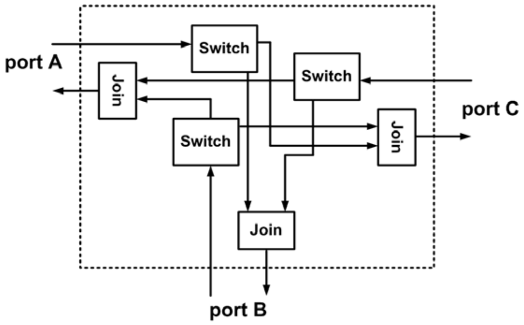

Fig. 4. Architecture of Router

2.2 *Network Topology*

The first key tenant for our network topology is the assumption that the the routing algorithm is simple. We choose the simplest algorithm possible: a binary decision at each router node. The data is now routed through the equivalent of a simple binary mux, and the next address calculation takes *no logic* – it is merely the rotation or swizzling of the data bits illustrated in Figure 5 for three pipeline stages. Routing now results with such low latency that it is beneficial to use single data word packets. There is now no need to have extra logic to calculate packet lengths, no need to set up routes beforehand, packet buffer reservation requirements are fully covered by the elastic protocol, and if a packet is blocked it doesn't have a tail that blocks other traffic, and there is no wasted traffic due to void packet filler words. This format does have the overhead of requiring address bits with every data word.

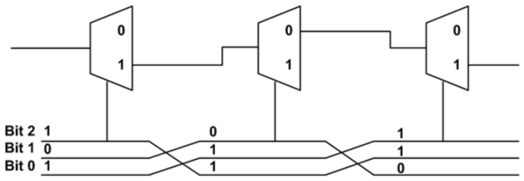

Fig. 5. Data Sequence Swizzling Scheme

The second key tenant is that the network is irredundant and cycle-free. Since there are no cycles, the network is deadlock-free and packets between a sender and receiver are guaranteed to arrive in order. Such topologies trade off additional routing nodes for simplified network routing overhead. One of the most efficient binary cycle-free topologies is a tree. For a balanced binary tree the maximum distance between any two hops is $\mathcal{O}(log_2\ n)$, giving a longest path of $2(n-1)$

routers for 2^n PEs. This number is important as it dictates the address overhead for each data word.

An addressing scheme is used that requires one address bit per router. This results in a sparse addressing scheme where $2n - 2$ bits are needed to address 2^n locally synchronous islands. One additional advantage to this topology is that the the source address is the "mirror" of the destination address. Hence, in any 2^n PE system where return addresses are required, this addressing scheme saves two data bits over a dense binary encoded address scheme. Therefore The additional cost in wires is offset by the simplicity of the routing algorithm and the equally important characteristic of obviating the need to include a source address or identifier in the packet. Note that this does not result in unique addresses for all IP in a system. For instance, the addresses for an 8 node system will require four address bits, where the leftmost node in the tree addresses the other nodes with bits $1xxx, 011x, 010x, 0011, 0010, 0001$, and 0000. This can be inferred from Table 1 which shows the method of calculating the return address from the received address by rotating and inverting the bits.

	IP1	**IP2**	**IP3**	**IP4**	**IP5**	**IP6**	**IP7**
Target Addr.	0xxx	100x	101x	1100	1101	1110	1111
Rec'd Value	xxx0	x100	x101	1100	1101	1110	1111
Reverse bits	0xxx	001x	101x	0011	1011	0111	1111
Return Addr.	1xxx	110x	010x	1100	0100	1000	0000

Table 1
Return Address Calculation to IP0 from IP1–7

3 Synchronizing Interface

We have designed two simple synchronizing interfaces, a clocked version and an asynchronous version, to allow us to compare relative efficiencies of the different approaches. We have also developed a low latency synchronizer that we employ with the asynchronous FIFOs.

3.1 Interface Circuit

The synchronizing interface in a GALS system is one of the primary areas of inefficiency in this design style. Since each PE can run at an independent frequency, data transmitted across the clock boundary must be synchronized. The overhead for synchronization can be substantial. These interfaces also require some sort of buffering to decouple the processing elements and network so that they can operate concurrently. Figure 6 shows the synchronizing interfaces between the network and a processing element (PE), with an independent buffering.

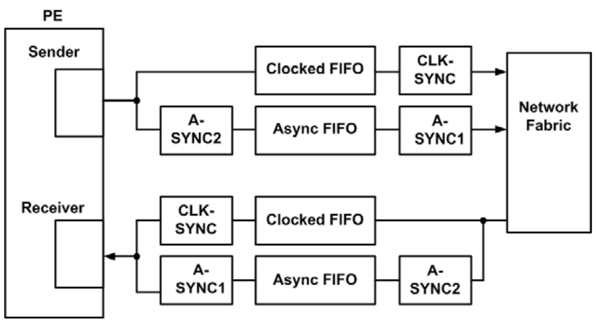

Fig. 6. Interface Circuit

3.2 Buffering FIFOs

Two FIFO designs are employed, a clocked design and asynchronous design. Both designs are simple four-deep linear shift FIFOs. The asynchronous FIFO has a significant advantage since data propagates very quickly from the tail to the head of the FIFO, whereas in the simple clocked domain it takes four full clock cycles. The clocked FIFO is implemented as four elastic half buffers.

The asynchronous FIFO is implemented using domino gates. This protocol is compatible with the synchronous phase based channel protocol we have used in this design, but some protocol conversion is necessary to convert from async handshake to a clocked interface.

3.3 Synchronizer and Channel Interfaces

The clocked synchronizer is implemented with two flops plus some additional logic to handshake between the domains to ensure correct data transfer. The full implementation, including the elastic buffers on each side of the synchronizer, is shown in Figure 7.

We have designed a novel "fast synchronizer" that uses mutual exclusion elements [9] rather than flops for synchronization. Additional logic is added to allow the fast synchronizer to communicate with pSELF modules, as shown in Figure 8. When coupled with the asynchronous FIFO, this circuit is able to send data and a valid signal into a clocked domain and receive an acknowledge signal in one clock cycle. Therefore this fast synchronizer has no synchronization penalty while the two-flop synchronizer generally consumes two clock cycles in each direction in the worst case.

Synchronizations must occur when crossing into a clocked domain. The clocked FIFOs use the same timing domain as their input port. Therefore, only one synchronizing interface is needed per clocked FIFO. The asynchronous FIFOs require synchronization on both ends of the FIFO since the valid or stall signals will be crossing into a clocked domain. This is a worst-case for the async FIFO. If an asynchronous network fabric were used, then only a single synchronizing interface into the PE clock domain would be needed. In figure 6, A-SYNC1 is a synchronizer for data and valid signal and A-SYNC2 is for the stall signal.

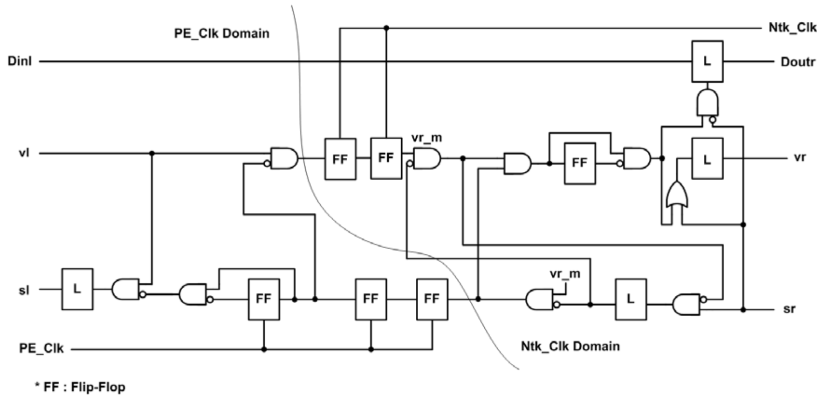

Fig. 7. Clocked Synchronizer

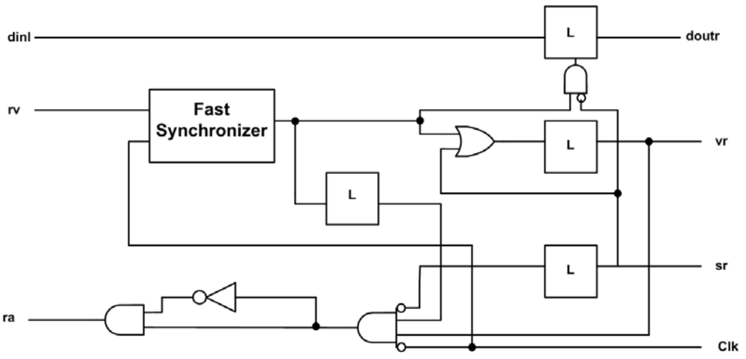

Fig. 8. Async synchronizer for valid and data

4 Chip Design

This paper reports simulation results from the chip design shown in Figures 9 and 10. The chip consists of a single processing element (PE), three synchronizing interfaces which are selected through muxes, a pSELF router, a pEHB, and three configurable ports. With this design we can simulate our SoC by interconnecting several chips. The network circuits and synchronizing interfaces on the chip share a clock, and the PE contains its own clock pin. This allows us to test the design with the PEs having an independent frequency island.

The chip design has four main components: 1) *Processing Elements* (or PEs) are designed to generate traffic, analyze the results, and store traffic data in local memory arrays as histograms. 2) *Network Interfaces* for synchronizing and buffering into the clock frequency of the PEs. 3) A *Network Fabric* that contains our pSELF channel protocols and logic, and consist of routers and phase elastic half buffers (pEHB). 4) *Scan Interface* allows us to configure the PEs and scan out the results of a network traffic evaluation. The PE's, network interfaces, and fabric all communicate via the pSELF protocols.

Space limitations on the chip prevented the design of a full SoC design, so multiple chips will be used to emulate such a network. We will only receive five chips from the foundry, so we have chosen a five node network as shown in Figure 11. This design will emulate a System on Chip (SoC) design with five independent frequency islands, and a network topology optimized for the statistical traffic patterns between the PEs. Routing through this topology is a binary decision in the network based on the value of the high order bit. A one value steers the packet to the left fork of the $\top$ and a zero value steers the packet to the right. These are indicated by 1 and 0 values in Figure 11 coming into the router node. The five PE design will be mapped to five chips as shown in Figure 12.

A PE acts as one intellectual property (IP) core. It can send or receive data to or from the network fabric. Sending and receiving can happen in the same clock cycle because we have concurrent bidirectional pSELF interfaces.

The PE implements three main functions: traffic generation, network data collection, and result scan out. This custom processor is necessary for us to run real-time network and interface patterns, and store the results from these runs. The PE works under three modes : scan in, run, and scan out. The PE must be configured to run a traffic simulation. This information is loaded through a scan chain. The PE is then reset and the traffic is driven on the network, while the PE's collect data on packet latency. The PE is then put into scan out mode to retrieve data collected from the network traffic simulation.

The data words for each communication in the test chip contain a payload that allows us to calculate the network latency. The data word consists of 9 bits containing a 3-bit address, 6-bit time stamp, and valid and stall bits. These small values are due to pin and area limitations on the chip.

As a sender, the PE can send up to one data word per clock cycle to the network interface. The Elastic protocol is maintained across the PE to network interface. Therefore, if the network fabric is congested or synchronization requires more than one cycle, the network asserts a stall signal to ask PE to stop sending data. When

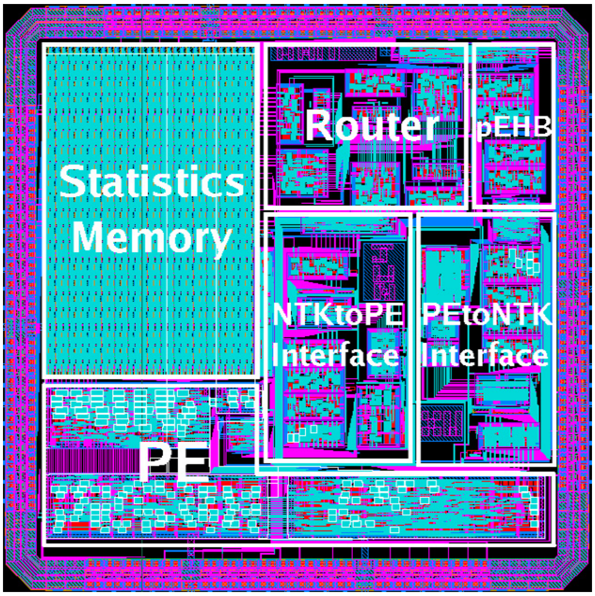

Fig. 9. Physical Design of Chip

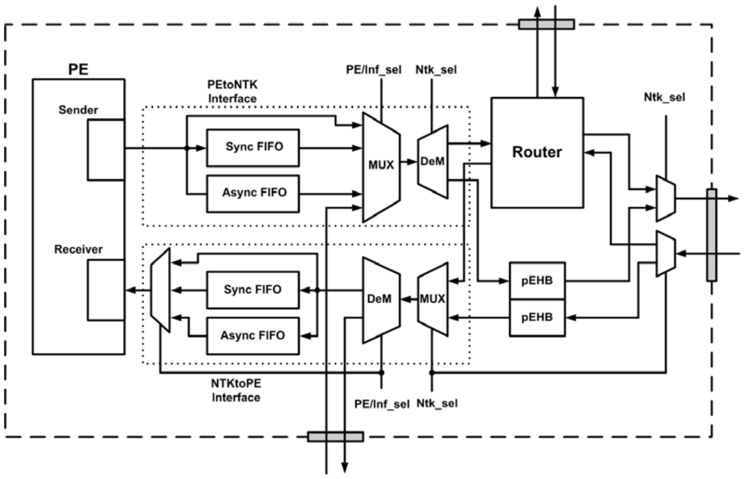

Fig. 10. One Chip Block Diagram

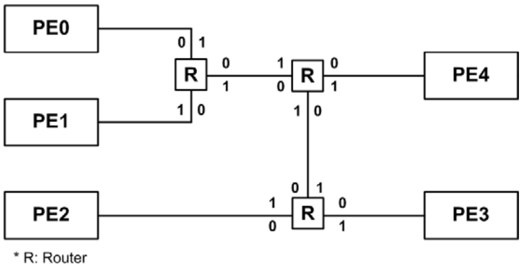

Fig. 11. Network Fabric Structure

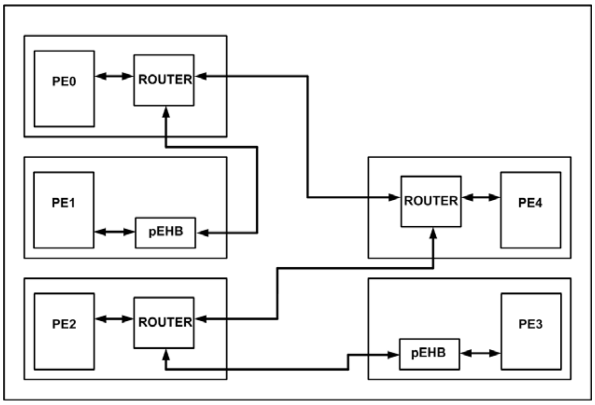

Fig. 12. Network Chip Architecture

sufficient space is available in the buffers, then the stall is retracted. The PE can be configured to transmit any programmable traffic pattern. The current timestamp is recorded whenever a packet is ready to be sent to the network. If the network has asserted a stall signal, the packet cannot be sent but the latency of the packet increases. The PE generates traffic to destination PEs in one of the following three modes:

(i) *Pseudo Random Generator*: A 3-bit linear feedback shift register (LFSR) is used to generate random 3-bit PE addresses. Each of these LFSRs can be loaded with an initial seed value. For simplicity, the initial value of pseudo random registers are reset to one.

(ii) *Shift Register Array*: An eight-deep three-wide shift register is loaded with an address sequence to generate a data transmission pattern and destination.

(iii) *Echo Mode*: In this mode, PE will respond to all incoming packets by sending an "echo" packet back to the sender. Destination address and time stamp are updated in this mode.

The PE can also receive one data word per clock cycle from the network interface. The PE can be programmed to have a particular stall pattern to mimic an IP core that cannot receive data every cycle. The PE maintains an independent results table for each of the four PEs with which it communicates. Upon receiving a data word from the network, the PE calculates the latency of the message across the network, by subtracting the send timestamp from the current timestamp. The packet latency is calculated by subtracting the six-bit timestamp in the packet from the current timestamp. The value is normalized by adding 64 to the result when negative.

The latency is stored in a user programmable histogram. This allows the efficiency of the network fabric and synchronizing interfaces to be measured and evaluated. The six-bit timestamp values allow delay variations of up to 64 cycles to be measured. The histograms contain up to 64 hits per bin, with eight total histogram bins. The partitioning of the histogram bins is programmable. There are seven six-bit registers that define the boundaries between the histogram slots. The latency value of the packet is compared against the seven boundary values to determine which bin to increment. To save time and space, each bin is a linear feedback shift register that is initialized to value 3F hex. The LFSR is shifted once to increment the bin. The pseudo random sequence is decoded into the binary result upon scan out.

5 Results

The circuit was built using a simple digital University of Utah static library. Some additional custom cells were designed as part of this project including domino gates for the asynchronous FIFOs and a mutual exclusion element for the fast synchronizer. The results reported in this section are ModelSim simulations of the extracted netlist of the circuit sent to fabrication. The simulations were made using a unit delay model since the custom gates were not characterized for timing.

The design contained 61K transistors, of which 33K were part of the data collection storage and 13K were associated with the network fabric. The maximum latency path consisted of 26 gate delays. At a FO4 delay of 200ps for this technology this will result in an expected performance of approximately 192MHz. The die area is 2400μm^2. For the simulation results reported here, the PE's were clocked at 50MHz and the network at 66MHz.

Two sets of results are of primary interest in designing this IC. A characterization of our elastic network controllers is necessary to automatically synthesize a network fabric. Secondly, a comparison between clocked and asynchronous interfaces is desired.

5.1 Characterization of Network Fabric

The chip was designed with an operational mode that completely bypasses the interface FIFOs. This permits the network to be characterized independent of the FIFO and nondeterministic delays of the synchronizers. In this mode the network and PEs are clocked with the exact same frequency. The results in this section all use the synchronous interface bypass mode.

The result in this section are all reported with the PE's configured to run under the same operational mode. Each run consisted of 500 messages. The PE's are configured to deliver and receive one packet every clock cycle. They are also configured to send packets to the same PE for the entire run. Hence for all of the tests in this section the links under test are operating at a 100% utilization factor. When the network is congested a stall is asserted and the PE must delay the delivery of the packet. In this case the latency is of the packet incremented for each stalled cycle.

The PEs in the chip cannot communicate directly with the IO pins, but must first pass through either an elastic buffer or router. The network topology is configured as shown in Figure 12. This gives six distinct classes of network paths between processing elements based on the type and number of elastic network nodes traversed. The buffering network nodes will be either an elastic buffer (E) or router (R).

Table 2 shows the latency result when applying network traffic between pairs of processors with all other PEs idle. Each class of paths had nearly identical results independent of the source and destination PEs. The table contains an example path for that network class. The latency through the network is effectively equal to the number of buffering elements, either routers or elastic buffers. This is as expected because the network is clocked. In an asynchronous version the delays will be much more dependent upon topology and the complexity of the buffering element.

Path Type	**# of Routers**	**Ntwk Elements**	**Path**	**Latency**
Class 1	1 Router	E-R	PE1 → PE0	2.96
Class 2	2 Routers	R-R	PE0 → PE4	2.96
Class 3	2 Routers	E-R-R	PE1 → PE4	3.94
Class 5	3 Routers	R-R-R	PE0 → PE2	3.94
Class 4	3 Routers	E-R-R-R	PE1 → PE2	4.93
Class 6	3 Routers	E-R-R-R-E	PE1 → PE3	5.91

Table 2
Network Fabric Latency

The next set of tests shown in Table 3 were designed to exercise the elasticity of the network operating under conditions of extreme congestion. Two or more PEs all delivered traffic to the same destination PE. Most of the network links are stalled due to the congestion in these conditions. The latency between the source and destination PE will be due to the percentage of bandwidth available to that PE. However, even in these extreme circumstances the worst-case end-to-end latency was reduced by only a factor of approximately 3.5 times the nominal delay. This worst case condition occurred on the longest path containing five buffering network elements. These results show that the elastic network contains a bounded worst-case latency. (This assumes that the receiving PE is always receptive in the high congestion mode.)

Concurrency	**Orientation**	**Aver.**	**Total Aver.**
2 Senders	PE0 → PE4	4.90	6.38
	PE2 → PE4	7.86	
3 Senders	PE0 → PE4	7.86	9.17
	PE1 → PE4	12.80	
	PE2 → PE4	6.86	
4 Senders	PE0 → PE4	11.78	15.23
	PE1 → PE4	18.68	
	PE2 → PE4	11.78	
	PE3 → PE4	18.68	
4 Senders	PE0 → PE3	13.75	12.23
	PE1 → PE3	20.90	
	PE2 → PE3	4.93	
	PE4 → PE3	9.35	

Table 3
Congestion of Signal

5.2 Comparison of Synchronization Style

In this section the PE and network fabric are clocked at different frequencies. All the PE's are operated at the same frequency, and the network fabric is operating at a different frequency than the processing elements. This requires synchronization between the network fabric and the PEs. For the results reported in this paper, the network was operating at a frequency 1.33× the PE frequency.

Three of the five PEs are configured to send traffic out onto the network. Each of these PE is configured to send packets out every other clock cycle, giving a 50%

activity factor. Two of the PEs send packets to a single destination node, and the third PE is configured to send data out to all PEs in a pseudo-random sequence. The PE's will receive packets each cycle without blocking.

Figure 13 shows the distribution of latencies when the PEs are configured to use the clocked or asynchronous FIFOs and synchronizing interface protocols. The asynchronous interface results in a substantially reduced latency compared to the clocked design. The increased performance is due to two factors: the FIFO and synchronizer latencies. The asynchronous FIFO is a simple four-deep flow-through FIFO. This simple efficient design allows data to propagate through the FIFO in approximately eight gate delays, which is much less than the clock cycle. The similar clocked implementation requires four cycles. We have also coupled a novel synchronizer we are developing with our asynchronous FIFOs. This new design can synchronize and handshake between the PE and asynchronous FIFO in a single clock cycle. In our simulations this results in nearly no cost for synchronization. The clocked protocol uses a typical two-flop synchronizer and associated protocol that requires three to four clock cycles of latency to synchronize and handshake between the two clock domains.

Figure 14 reports the average latency when the data is compiled based on three different network traffic patterns. In these three cases the two PEs communicating with the fixed destination will send the messages through one, two, or three router nodes. The third PE continues to send data to a pseudorandom destination. This table indicates that the increased overhead between the clocked and asynchronous network interfaces can mostly be attributed to the synchronization delays with some associated congestion produced by the stalling.

These result motivate a more detailed study of synchronizing interfaces. While the clocked interface can be substantially improved, these results indicate that there may always be a substantial advantage to some form of asynchronous synchronizing interface.

6 Conclusion

This paper reports on a test chip that implements a new latency insensitive protocol. The chip implements the new pSELF protocol applied to network applications

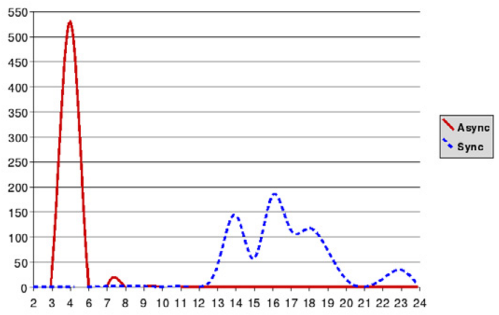

Fig. 13. Latency when using asynchronous versus clocked synchronizing interfaces

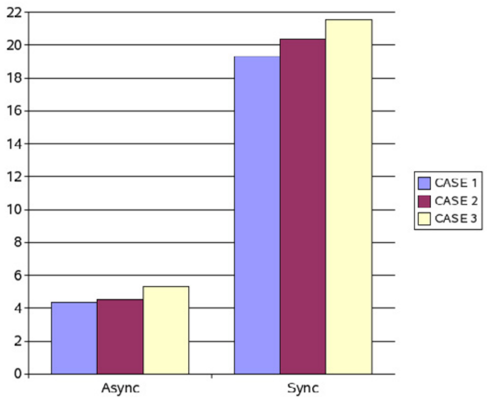

Fig. 14. Average Latency

and interfacing to clocked processing elements. The chip also contains the design of a novel synchronizer that is coupled with asynchronous FIFO. Results are presented from simulations of the extracted netlist. The network elements consist of buffers and routers. Both these elements show predictable performance, which enables a synthesized network flow. Further, these lightweight network components behave extremely well under congestion with predictable worst-case latency. A comparison is made between a simple traditional synchronizing FIFO and our new fast synchronizer and asynchronous FIFO. Total network latency in this example is reduced by a factor of four or more with the new synchronizing interface. This chip has been taped out the results reported here will be performed on silicon when the chips return.

References

[1] Campobello, G., M. Castano, C. Ciofi and D. Mangano, *GALS Networks on Chip: A New Solution for Asynchronous Delay-Insensitive Links*, in: *Design, Automation and Test in Europe*, 2006, pp. 1–6.

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

[3] Carloni, L. P. and A. L. Sangiovanni-Vincentelli, *Coping with latency in SOC design*, IEEE Micro **22** (2002), pp. 24–35.

[4] Cortadella, J., M. Kishinevsky and B. Grundmann, *Synthesis of synchronous elastic architectures*, in: *Proceedings of the Digital Automation Conference (DAC06)*, IEEE, 2006, pp. 657–662.

[5] Dobkin, R., R. Gionsar and C. P. Sotiriou, *Data synchronization issues in GALS SoCs*, in: *International Symposium on Asynchronous Circuits and Systems*, 2004, pp. 170–179.

[6] Gebhardt, D. and K. S. Stevens, *Elastic Flow in an Application Specific Network-on-Chip*, in: *Third International Workshop on Formal Methods in Globally Asynchronous Locally Synchronous Design (FMGALS 07)*, Elsevier Electronic Notes in Theoretical Computer Scinece, 2007.

[7] Ginosar, R., *Fourteen ways to fool your synchronizer*, in: *International Symposium on Asynchronous Circuits and Systems*, 2003, pp. 89–96.

[8] Jacobson, H. M., P. N. Kudva, P. Bose, P. W. Cook, S. E. Schuster, E. G. Mercer and C. J. Myers, *Synchronous interlocked pipelines*, in: *International Symposium on Asynchronous Circuits and Systems*, 2002, pp. 3–12.

[9] Seitz, C. L., *Ideas about arbiters*, Lambda **1** (1980), pp. 10–14.

[10] Semeraro, G., G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas and M. L. Scott, *Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling*, in: *Symposium on High Performance Computer Architecture*, 2002, pp. 29–42.

[11] Singh, M. and M. Theobald, *Generalized Latency-Insensitive Systems for GALS Architectures*, in: *Formal Methods in Globally Asynchronous Locally Synchronous Design (FMGALS 03)*, 2003.