Serialized Asynchronous Links for NoC

S. Ogg[1], E. Valli[2], B. Al-Hashimi[1], A. Yakovlev[3], C. D'Alessandro[3], L. Benini[2]
University of Southampton[1], University of Bologna[2], Newcastle University[3]
so04r@ecs.soton.ac.uk, bmah@ecs.soton.ac.uk*

Abstract – *This paper proposes an asynchronous serialized link for NoC that can achieve the same levels of performance in terms of flits per second as a synchronous link but with a reduced number of wires in the point to point switch links and reduced power consumption. This is achieved by employing serialization in the asynchronous domain as opposed to synchronous to facilitate the removal of global clocking on the serial links. Based on transistor level simulations using 0.12 µm foundry models it has been shown that it is possible to achieve the same level of performance as synchronous but with 75% reduction in wires and 65% reduction in power for a 300 MFlit/s link with 8 buffers with a switch clock speed of 300 MHz. Furthermore the paper presents the design requirements arising from interfacing switches of synchronous NoC and asynchronous serial links.*

Keywords: Network-on-Chip, Serial, Asynchronous, Point-to-Point Links.

I. INTRODUCTION

As multiprocessor system-on-chip solutions increase there are benefits to provide a scalable on chip communication architecture. One promising approach is Network-on-Chip (NoC). The growth of research into NoC has led to a number of viable architectures, examples [1-5]. Typically the NoC consists of network interfaces which allow a core to interface to the network, switches which are responsible for routing the packet and links which connect the switches together. Numerous NoC architectures adopt a synchronous approach and more recently there have been studies of asynchronous NoC [6, 7] which highlight some of the problems with synchronous NoC such as global clock power consumption, clock skew and electro-magnetic interference. An asynchronous point-to-point link that can be used for communication has been investigated in [8]. This scheme uses clock pausing techniques to pass data from the synchronous to asynchronous domains. Power in synchronous design can be reduced by lowering the clock speed, but to maintain the throughput the data width would need to be increased by the same factor. Interconnect cost, in terms of the number of wires required between switches, could grow to be considerable in NoC if the data width is increased since each switch is effectively connected by a point-to-point link to a neighboring switch and the high cost of parallel links has been shown in [9] which compares fully parallel and bit-serial buffered wires. It is expected with further scaling down of technology the number of point-to-point links between the switches of a NoC will grow as more and more cores are integrated into a system.

This paper proposes the application of serialization as a means of reducing the number of wires between NoC switches. Byte-level serialization of the data is performed as opposed to a fully bit-serial single wire link. Furthermore, the serialization is employed in the asynchronous domain to remove the need for high frequency global clocking of the serial links which would be required in a synchronous design. The paper is organised as follows, section II provides the motivation, section III describes the asynchronous link and circuits, section IV discusses word-level acknowledgement for increasing the performance, section V is the experimental results and finally VI the concluding remarks.

II. MOTIVATION

Synchronous NoC allows for high a throughput of data due to the pipelining of the data path where the switches and the wire pipelining buffers are clocked together [2]. In a single link (Fig. 1a) the two switches are connected together with a wire segmented by a series of synchronous clocked buffers. A high speed global clock is attractive to allow a high throughput between the switches. However, high speed clocks may have problems such as skew, timing closure and power dissipation. A slower clock could be used to alleviate these problems [10] but the throughput would be decreased. One way to increase the bandwidth of a slow clocked system would be to make the data path wider but in NoC a wider data path would mean an increase in the number of wires in the point to point links, increasing the wiring area and routing complexity considerably.

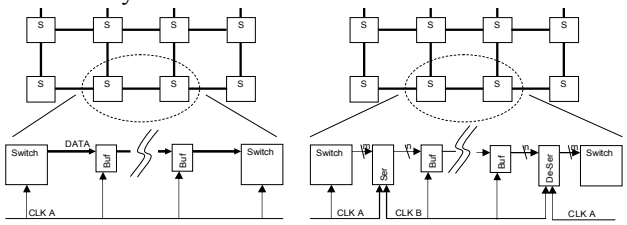

Fig. 1 NoC with Synchronous Link

In a slow clocked synchronous NoC with wide data paths the number of wires between the point to point links

* The authors would like to acknowledge the Engineering and Physical Sciences Research Council (EPSRC) for funding under grant no. EP/C512804.

can be reduced through serialization as it is being proposed in this work. Consider a simple serialization scheme (Fig. 1b) the number of wires required would reduce from the original *m* to the reduced *n*. However, this would also mean that the 2nd clock (CLK B) driving the serializer, de-serializer and wire-buffers would need to be introduced. CLK B would need to be *m*/*n* times faster which could mean a 2nd clock tree spanning the chip area covering the NoC. Also, if no FIFO or clock pausing mechanisms are used to pass data between the two clock domains the two clocks would need to be tightly phased locked each other and CLK B would need to be an integer value times faster than CLK A in order that no timing violations occur when data or control signals pass between the two domains.

A way around this is to serialize in the asynchronous domain so that a single slow global clock is maintained for the switches and the serialized data path between the switches allows for same throughput with a reduced number of wires. The introduction of asynchronous elements to the link would allow a structure as shown in Fig. 2. The switch would interface directly to a synch/asynch interface and then go through an asynchronous serializer. The benefit of this approach is that the data is serialized and thus saves wire area but also does not require a second higher speed clock to be fed into the serialization circuits and to the wire-pipeline buffers. It should be noted that the employment of serialization in the context of NoC has been proposed to reduce energy consumption[11].

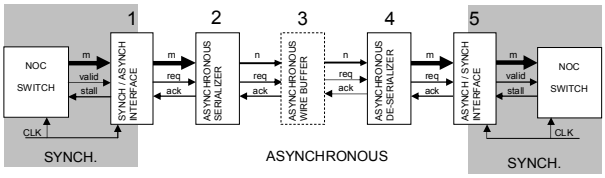

Fig. 2 Proposed Serialized Asynchronous Link

III. ASYNCHRONOUS LINK DESCRIPTION

The asynchronous link consists of a synch/asynch interface, serializer, wire buffers, de-serializer and an asynch/synch interface. Additional buffers can be inserted to maintain performance if needed over long wire lengths. Circuits have been designed for the implementations of the synchronous to asynchronous interfaces and the serializer and de-serializer. The design of each of the modules will be described in the corresponding sub-section. The asynchronous point-to-point link circuitry is implemented using standard logic cells and two common asynchronous cells, the C-Element[12] and the David-Cell[13], Fig. 3. A 4 deep FIFO was used in the synchronous to asynchronous interface and asynchronous to synchronous interface to give a total of 8 possible spaces for data along the link, the same as the synchronous link. The presented work shows a proof-of-concept implementation of an asynchronous link using a bundled-data link.

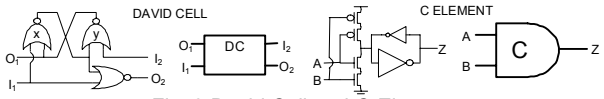

Fig. 3 David Cell and C-Element

Synch/Asynch/Synch Interface

The synch/asynch interface (Fig. 4) is basically a FIFO with a synchronous side that can write and an asynchronous side that can read. The FIFO is 32 bits wide and 4 registers deep. A FIFO is used to effectively break the dependency of the asynchronous side from the synchronous side. The synchronous side has four registers which are synchronously written to when the appropriate WR_EN(x) signal is active. For each register there is an associated flag, the flag consists of two clocked D-Type flip flops. The use of two flip-flops to build a synchronizer is known to ensure protection against metastability [14]. The flag can be asynchronously cleared by using CLEAR(x) which is gated with the asynchronous reset attached to the D-Type. The VALID and STALL signals are used to determine if there is space for the data on FLITIN to be written into one of the registers. The chain of David-Cells effectively form a 1-hot sequencer where one of them is always active. The C-Elements control the request and acknowledge handshaking and trigger the David-Cells in sequence. The asynch/synch interface (
Fig. 5) is similar to the synch/asynch interface and has an asynchronous latch writer and synchronous latch reader.

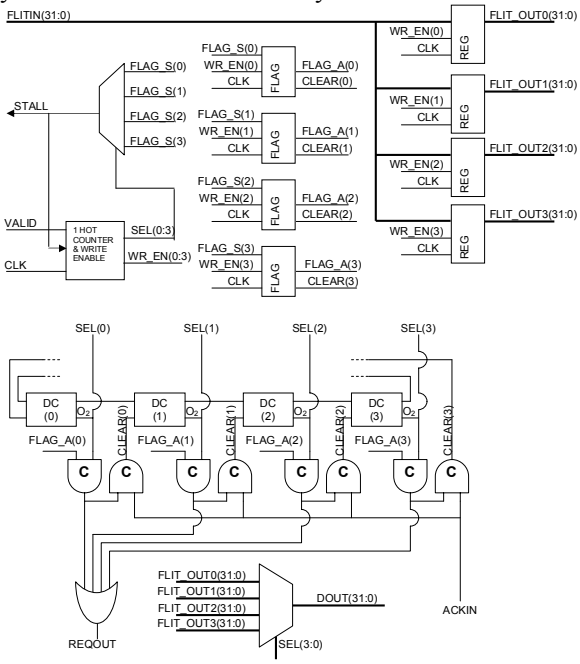

Fig. 4 Synchronous to Asynchronous Interface

Asynch. Serializer, De-Serializer and Wire Buffer

The asynchronous serializer (Fig. 6a) consists of several David-Cells which select each 8 bit slice of the 32 bit data in turn. At reset the output O_2 of DC(0) is logic '1' and output O_2 of DC(1-3) are logic '0'. The REQIN signal gated with SEL(0) triggers the start of the REQOUT / ACKIN sequence which is performed 4 times, each time

the next 8 bit slice of the 32 bit data word is selected and latched at the output. The circuit can easily be modified to serialize less and break the 32 bit word in larger slices by decreasing the number of David-Cells and making the data path DOUT wider.

The asynchronous de-serializer (Fig. 6b) takes 4 slices of 8 bits and re-constructs the original 32 bit data. At reset the output O_2 of DC(0) is logic '1'. REQIN will go high signifying the first 8 bit slice is valid on DIN. The output of the C-Element LE(0) will then trigger and go high and latch the 8 bit slice into place. The REQIN/ACKOUT cycle is repeated 4 times until the 32 bit word is re-built and then the REQOUT is taken high to signify to the next stage the valid 32 bit data is ready. The circuit can be altered for larger or smaller slice widths by reducing or increasing the number of David-Cells in the chain and altering the data path width.

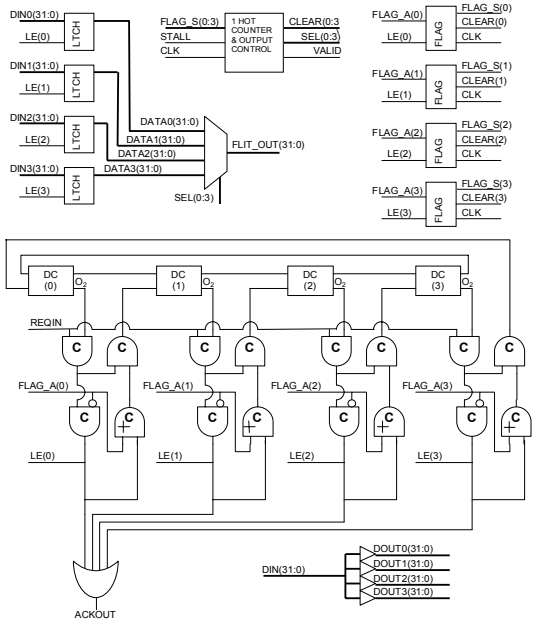

Fig. 5 Asynchronous to Synchronous Interface

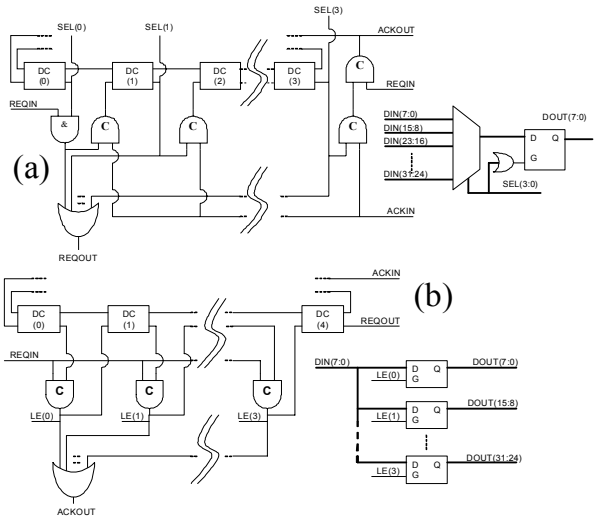

Fig. 6 Asynchronous Serializer/Deserializer

The asynchronous wire buffer is based on a simple four phase latch control circuit [15]. It essentially latches the data on the falling edge of REQIN. The C-Element regulates the request and acknowledge handshaking safely. One point to note about this circuit is that the REQIN/ACKOUT side is not fully de-coupled from REQOUT/ACKIN side. If several of the wire-buffers are chained together then at best only every other buffer in the chain will be in use at a time. This does not present a problem in our case as the wire-buffering is a mechanism for transporting data rather than storage.

IV. Asynchronous Acknowledgements

One of the problems associated with a per-transfer acknowledgement is the need for the receiver or line buffers to acknowledge every transfer. As the parallel data gets more and more serialised the number of request-acknowledge cycles per word increases. One possible way around this is to use a coarser grain acknowledgement that acknowledges at the word level. Word level acknowledgement does have some implications such as timing closure at the receiver which must be able to receive multiple transfers correctly and the need for some self regulated timing mechanism, such as a clock, at the transmitter to space the burst transfers out such that there are no timing violations incurred at the receive end. The proposed link can accommodate two types of acknowledgements, per-transfer and per-word. Fig. 7 shows the proposed link with word level acknowledgement by modifying the serializer, de-serializer and wire buffer.

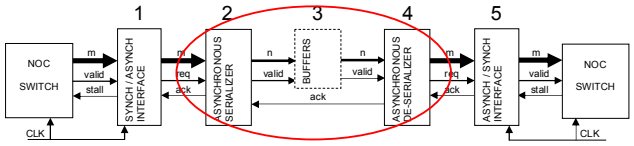

Fig. 7 Serial Asynchronous word-level acknowledgement

The buffers along the length of the wire can be replaced by simple buffers or an even number of invertors. The serializer (Fig. 8a) uses a multiplexer with each slice of a word being selected in turn. The VALID signal goes high when there is valid data on DOUT and signified to the receiver end that the data can be used. The VALID signal goes high 4 times, once for each slice of the word. The timing of the VALID signal is derived from the ring oscillator constructed by 5 back to back invertors. To adjust the frequency of the best the number of invertors can be altered or different sizes can be used depending upon requirements. To ensure that VALID only goes high when the DATA is valid the respective timing between DATA and VALID can also be tuned by selecting different taps off the ring oscillator if necessary. Furthermore, if tolerance becomes problematic the VALID signal generation can be combined with the SELect signals to increase robustness.

The de-serializer (Fig. 8b) employs a shift register. This was done to see the effects of a shift register based de-serializer versus the original mux based de-serializer. The

data is shifted in on DIN every time VALID goes high and the data slices are serially shifted onto DOUT. At the same time a single bit pulse is shifted down a single bit shift register of the same length to provide a REQOUT signal to the next asynchronous block to inform it the whole word has been built and is valid. ACKIN clears the single bit shift registers and removes REQOUT completing the handshake.

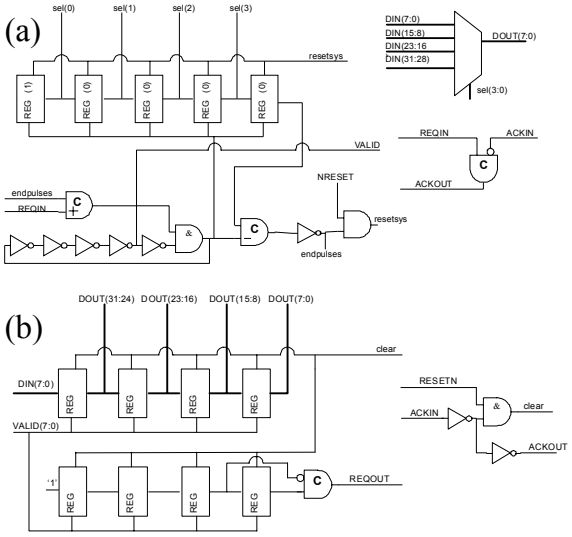

Fig. 8 Word Level Serializer/Deserializer

V. EXPERIMENTAL RESULTS

To validate the performance, power consumption and area overhead of the proposed serialized link, three links were synthesized using 0.12 µm and simulated using Cadence Spectre. The three links (Fig. 9) are: a fully synchronous link with no serialization (I1), proposed asynchronous link with per-transfer acknowledgement (I2) and per-word acknowledgement (I3). Note four buffers were used in each link and in the case of the serial links has 8 bit data.

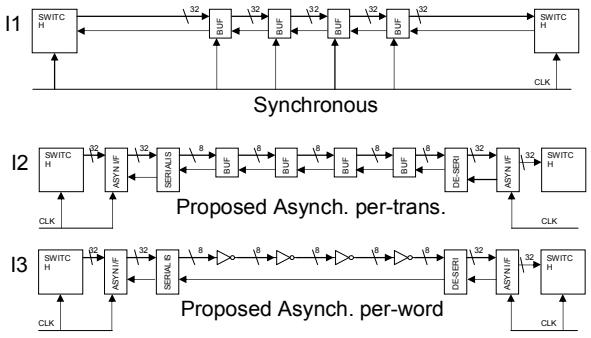

Fig. 9 Simulated Implementations

Fig. 10 shows the number of wires needed to achieve a certain bandwidth across a link. The synchronous link with 100, 200 and 300 MHz clock speeds are shown with the proposed link. As is seen the number of wires increase dramatically in the synchronous link as bandwidth increases. The number of wires for the proposed asynchronous serial link remains constant independent of the switch clock speed as the asynchronous link is not reliant on a synchronous clock to transfer data along the wire. Fig. 10 shows that it is possible to achieve the same performance as the synchronous link but with less wires. For example, the proposed link (I3) can support 300 MFlits/s using a 300 MHz switch clock with 8 wires whereas the synchronous link (I1) would need 32 wires at 300 MHz which is a 75% reduction. It is interesting to note that the number of wires in the synchronous link would need to increase if the switch clock speed was reduced from 300 MHz to 100 MHz and maintain the same throughput, this would require an increase to 96 wires at 100 MHz.

To give insight into the wiring area for a given wire length consider Fig. 11. The benefit of reducing the number of wires can clearly be seen, especially for longer wire lengths. For example, assuming a wire length of 1000 µm, I3 has a wiring area cost of approximately 7,500 μm^2 whereas the synchronous link (I1) is approximately 30,000 μm^2. As the wire length increases there is a large increase in the area cost for the synchronous link, unlike the proposed asynchronous link which has moderate increase. Note Fig. 11 was produce using the following equation:

$$AREA_{DataWires} = L \times (N \times Met_W + (N+1) \times Met_G),$$

where N is the number of wires, L is the length of the wires, Met_W is the minimum metal width and Met_G is the minimum metal gap. For the global METAL6 layer in the ST 0.12 µm technology Met_W = 0.44 µm and Met_G = 0.46 µm.

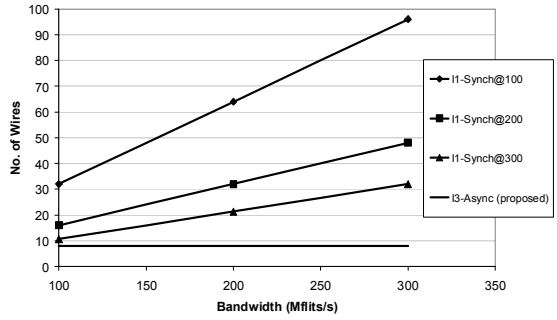

Fig. 10 Bandwidth vs. Wires

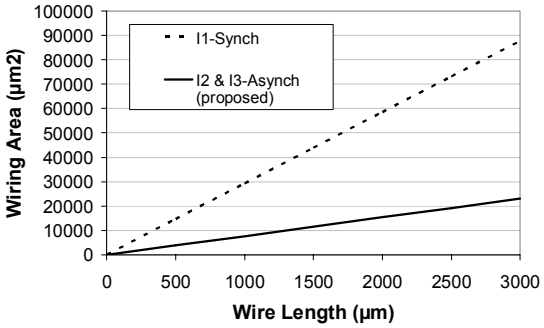

Fig. 11 Wire Area

The power consumption of the synchronous and the proposed asynchronous link are shown in Fig. 12 with switch clock speed of 100 MHz for different numbers of buffers in the link. As expected when a small number of buffers are used, such as 2, the synchronous

implementation uses less power compared to the asynchronous due to the extra overhead of the synch/asynch converters and serializers. However, when the number of buffers increase the power in the synchronous implementation increases unlike the asynchronous implementation which remains relatively the same. Comparing 2 buffers against 8 buffers for the wire link it can be seen the that power for the synchronous implementation (I1) increases 300% from 372 µW to 1498 µW which is expected since there is four times the number of synchronous buffers. The asynchronous per-transfer scheme (I2) shows a small power increase of 20% of the 589 µW to 712 µW, while the per-word acknowledgement scheme (I3) shows the least power increase of 2%, 623 µW to 637 µW, due to invertors being used along the length of the wire instead of latched buffer elements. Similar power consumption results can be obtained when the switch clock speed is increased to 300 MHz (Fig. 13). As expected the synchronous link power increases with clock frequency and it can be seen that power increases from 1498 µW to 3229 µW for 8 buffers. The best power saving is obtained when the switch clock speed is 300 MHz and the number of buffers is 8, power is reduced by 65% from 3229 µW to 1110 µW when going from synchronous to asynchronous in this case.

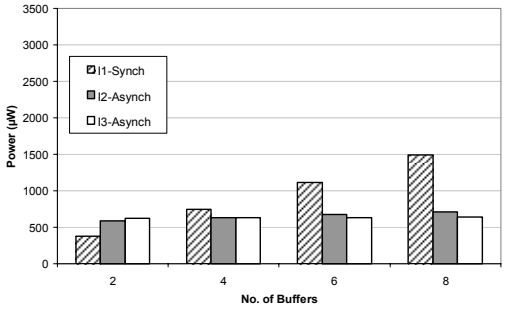

Fig. 12 Number of Buffers vs. Power @ 100 MHz

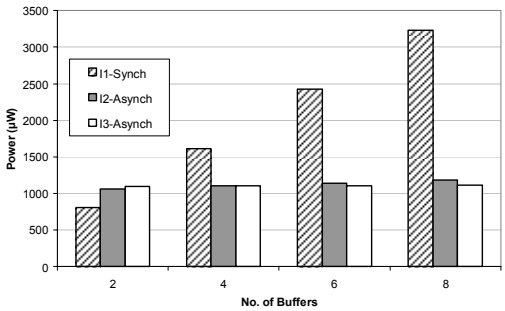

Fig. 13 Buffers v Power @ 300 MHz

To give insight as to where the power consumption is in the various components of the links, Fig. 14 shows a breakdown of the power for 50% usage. It can be seen that the dominant power in the asynchronous implementations (I2 and I3) are the asynch/synch and synch/asynch conversion circuits. This is expected since these circuits contain clocked synchronous parts. Comparing the proposed asynchronous links I2 and I3 which serialize down to 8 bits, it can be seen that that overall power used is similar. The I3 buffer power is considerably smaller than I2 at 9 µW versus 82 µW due to the fact that the buffers are simple invertors along the length of the wire and not latched elements as is the case for I1 and I2. The I2 de-serializer uses more power than the I3 de-serializer as a shift register based implementation is used instead of a de-multiplexer, so all four registers are being latched every time a slice of the flit arrives opposed to just one register being latched in the de-multiplexer version.

The average power was obtained for the transfer of 4 data items (0xA5A5A5A5, 0x5A5A5A5A, 0xA5A5A5A5, 0x5A5A5A5A) which exercise the data wires as much as possible and give worst case data activity. The time the link is in use when transferring the 4 data items is approximately 70 ns on the synchronous implementation running at 100 MHz, and the simulation runs set to 140 and 280 ns. This allows the average power for 50% usage to be obtained. The link can be considered 'in use' when one or more of the buffers is occupied by a flit/data. The same simulation run time was used for the asynchronous implementations to provide a comparison between the implementations. The power for each block was obtained through Spectre simulations, the average of the supply voltage multiplied by the current over the simulation run time was taken.

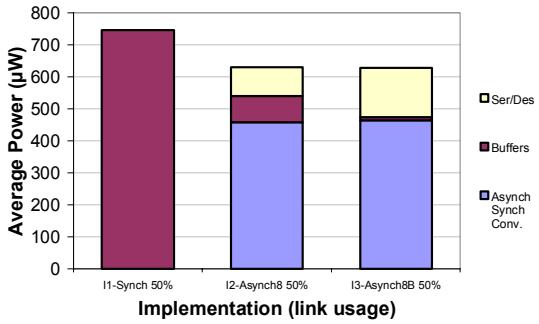

Fig. 14 Average Power for 50% usage

The area overhead of the synchronous and proposed asynchronous links are given in Table 1. To give an idea of which portions of the asynchronous link use most resource a breakdown of the circuit or cell area used for each module for the implementation I2 is shown in Table 2. The proposed architectures, I2 and I3, have an area increase of approximately 20% compared to the synchronous link (I1).

Table 1 Area overhead of the synchronous and proposed link

Implementation	Area (µm2)
Synchronous (I1)	15864
Asynchronous per-transfer ack. (I2)	19193
Asynchronous per-word ack. (I3)	18396

Table 2 Breakdown of Implementation I2

Module	Area (μm^2)	Qty.
Synch to Asynch interface	9408	1
Asynch 32 to 8 serializer	869	1
Asynch 8 wire buffer	294	4
Asynch 8 to 32 de-serializer	1030	1
Asynch to Synch interface	6710	1
Total	19193	

To evaluate the accuracy of the per-transfer and per-word performance we developed two equations which can be used to calculate the cycle delay of a word transfer and find the upper bound of the throughput. For the per-transfer acknowledge scheme (Fig. 15) the cycle delay can be calculated:

$$D = 4\times(4\times Tp + Treqreq + Treqack + Tackack + Tackout) + Tnextflit$$

where Tp is the propagation time along the wires (of which there are 4), Treqreq is the time of the request to write data into the buffer to the request to write the data out to the next buffer, Treqack is the time to request to write data into the buffer to the acknowledgment of the data, Tackack is the acknowledgement into the buffer to the acknowledgement out to the previous buffer and Tackout is the acknowledgement into the buffer to the output of a new slice of data. This is multiplied by 4 since the 32 bit flit is sent 8 bits at time and will take 4 transfers to complete a whole flit. Tnextflit is the time taken to get the next flit to be ready on the outputs of the transmitter.

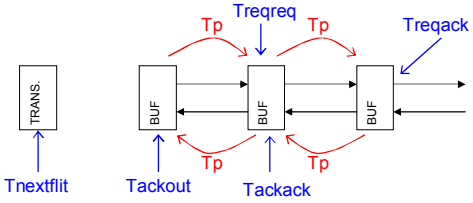

Fig. 15 Cycle Delay for the Per-transfer

For the per-word acknowledge scheme (Fig. 16) the cycle delay can be calculated using:

$$D = 10\times Tp + 8\times Tinv + Tvalidwordack + Tackout + Tburst$$

where Tp is the wire propagation delay (in this case there are 10), Tinv is the inverter gate delay (of which there are 8), Tvalidwordack is the delay from receiving a valid word to acknowledge output, Tackout is the acknowledge in to new flit output and Tburst is the burst period of the 4 slices of flit.

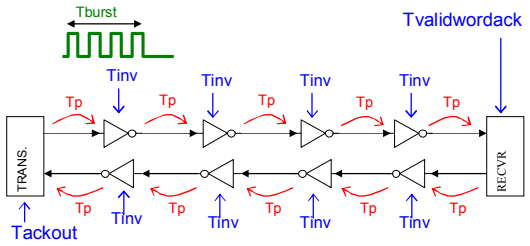

Fig. 16 Delay for per-transfer and per-word

The per-word equation can be checked using an example. Consider, Tp=0 since the simulation was gate level, Tinv=0.011 ns from the ST 0.12 CORE9GPLL datasheet, Tburst ~ 1.1 ns from simulation, Tvalidwordack ~ 0.7 ns and Tackout ~ 1.4 ns also from simulation. Using these values the per-word delay is 3.21 ns from which we obtain an upper bound throughput of around 311 MFlits/s which matches with the supported bandwidths shown in Fig. 10. Further improvements to the upper bound throughput could be achieved by earlier acknowledging or nacking which the authors are investigating for future work.

VI. Concluding Remarks

This paper has proposed and demonstrated the effectiveness of serialization in reducing the number of wires without compromising the performance. The potential problems with synchronous design such as global clock distribution and clock skew have also been reduced. The proposed asynchronous link also reduces power by up to 65% compared to the synchronous link when 8 buffers are used. Furthermore, we have compared the area overheads of synchronous and the proposed asynchronous link and shown that although the proposed link has a 20% circuit overhead the number of wires has been reduced by up to 75%.

The validations and comparison were carried out using synthesized gate level designs and realistic simulation environment. It is hoped the proposed link makes a valuable contribution to the area of efficient NoC architecture for multi-processor SoC.

REFERENCES

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: a scalable, packet switched, on-chip micro-network," in *DATE 2003*.

[2] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits and Systems Magazine,* vol. 4, pp. 18-31, 2004.

[3] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers,* vol. 22, pp. 414-21, 2005.

[4] D. Siguenza-Tortosa and J. Nurmi, "Proteo: a new approach to network-on-chip," in *IASTED Conference on Communication Systems and Networks*, Malaga, Spain, 2002, pp. 355-9.

[5] D. Wiklund and L. Dake, "SoCBUS: switched network on chip for hard real time embedded systems," in *IPDPS 2003*.

[6] M. Amde et al, "Asynchronous on-chip networks," in *System-on-Chip: Next Generation Electronics*, B. M. Al-Hashimi, Ed.: IEE, 2006, pp. 625-52.

[7] E. Beigne et al, "An asynchronous NOC architecture providing low latency service and its multi-level design framework," in *11th IEEE International Symposium on Asynchronous Circuits and Systems*, 2005.

[8] S. Moore, G. Taylor, R. Mullins, and P. Robinson, "Point to point GALS interconnect," in *International Symposium on Asynchronous Circuits and Systems*, 2002.

[9] A. Morgenshtein et al, "Comparative analysis of serial vs parallel links in NoC," in *International Symposium on System-on-Chip* Tampere, Finland, 2004, pp. 185-8.

[10] A. Pullini et al, "NoC Design and Implementation in 65nm Technology," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, 2007, pp. 273-282.

[11] L. Kangmin, L. Se-Joong, and Y. Hoi-Jun, "Low-power network-on-chip for high-performance SoC design," *IEEE Transactions VLSI Systems,* vol. 14, pp. 148-60, 2006.

[12] D. E. Muller and W. S. Bartky, "A Theory of Asynchronous Circuits," in *Proceedings of an International Symposium on the Theory of Switching*, 1959, pp. 204-243.

[13] R. David, "Modular design of asynchronous circuits defined by graphs," *IEEE Transactions on Computers,* vol. C-26, pp. 727-737, 1977.

[14] L. Morin and H. F. Li, "Design of synchronisers: a review," *IEE Proceedings E (Computers and Digital Techniques),* vol. 136, pp. 557-64, 1989.

[15] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Transactions VLSI Systems,* vol. 4, 1996.