Link Pipelining Strategies for an Application-Specific Asynchronous NoC

Daniel Gebhardt
University of Utah
School of Computing
gebhardt@cs.utah.edu

Junbok You
University of Utah
Dept. of Electrical and Computer Engineering
jyou@ece.utah.edu

Kenneth S. Stevens
University of Utah
Dept. of Electrical and Computer Engineering
kstevens@ece.utah.edu

ABSTRACT

Wire latency across the links of a NoC can limit throughput, especially in deep submicron technology. Stateful pipeline buffers added to long links allow a higher clock rate, but this wastes resources on links needing only low bandwidth. In asynchronous (clockless) NoCs, link pipelining can be done to only those that will benefit from both increased throughput and buffering capacity, and is especially useful in heterogeneous embedded SoCs. We evaluate two strategies that determine where link pipeline buffers should be placed in the topology. The first compares available link bandwidth, based on physical wirelength, to the throughput needed by each source-to-destination path, for each link. The second adds buffers to a link such that its bandwidth is at least equal to the throughput of a core's network adapter. These strategies were integrated into our network optimization tool for an application-specific SoC. Simulations were based on its expected traffic patterns, floorplan-derived wirelength, and uses self-similar traffic generation for more realistic behavior. Results show improved large-message network latency and output buffer delay of the network adapter. There was a slight power increase with the addition of pipeline buffers, but our proposal is a *complexity-effective* improvement by the power*latency product metric. The results indicate the strategy of pipelining certain links provides more efficiency opposed to a ubiquitous addition of buffers.

Categories and Subject Descriptors

B.4.3 [**Input/Output and Data Communications**]: Interconnections—*Asynchronous/synchronous operation*

1. INTRODUCTION

A link of a network-on-chip (NoC) can be *pipelined* using latch or register-based buffers when its wire delay is a limiting factor in throughput. This often occurs with long links, small process technology, and relatively fast clock speeds. Another benefit to link pipelining comes from the additional buffering space it provides, assuming a compatible link-level flow-control. This paper explores the system-level effects of link pipelining for an asynchronous (clockless) NoC that has already been optimized for link-specific bandwidth requirements.

Our work focuses on a class of system-on-chips (SoC) termed *application-specific* in which the NoC is designed and optimized for performance and energy qualities with knowledge of the specific tasks done by the SoC. These fixed-function SoCs can use many specialized cores and are less flexible than general-purpose chip-multiprocessors or even platform-SoCs. However, knowledge of the traffic, such as bandwidth between particular cores, provides an opportunity for optimization.

Many globally-asynchronous locally-synchronous (GALS) interconnect solutions rely on a clock, either with standard synchronous clock distribution, or a mesochronous method. However, an asynchronous network has a number of potential advantages over a clocked network in a GALS environment. Standard arguments for asynchronous (async) circuit design include robustness to process/voltage/temperature variation, and average-case instead of worst-case performance. However, there are also NoC-specific considerations. In a synchronous NoC, the clock tree for all routers and pipeline buffers can consume significant power as shown in a heterogeneous network [1], and in a large CMP (chip multiprocessor) 33% of router power [2]. Many SoC designs have quite bursty and "reactive" traffic. In this case, async methods are beneficial in that they consume little dynamic power during periods of low traffic without relying on clock gating techniques.

Available bandwidth on each async link is dependent, in part, on wirelength between sender and receiver. This is in contrast to clocked networks that commonly use a single frequency for all routers, which is wasteful on links not requiring high bandwidth. Bandwidth can be modified in an async NoC by changing the distance between routers, or by inserting link pipeline buffers on only those links that will benefit. We explore this concept here, with an evaluation of insertion strategies for heterogeneous SoC designs, and offer another knob to turn for the NoC engineer for power–performance trade-offs.

Figure 1 shows the functional components of an async NoC, and how it interfaces with the cores of an SoC. A core has an interface using some type of standard protocol, such as OCP or AMBA. A network adapter converts this protocol to the one used in the particular NoC design. It can also synchronize between two timing domains; in our work, between the core's clock domain and the asynchronous NoC domain. This also enables each core to operate at its own frequency, as is done in globally-asynchronous locally-

NOCS '11 May 1-4, 2011 Pittsburgh, PA, USA

synchronous (GALS) design. The routers are entirely async and do not use a clock to communicate to each other, or to the network adapters. An example implementation of this is presented in [3].

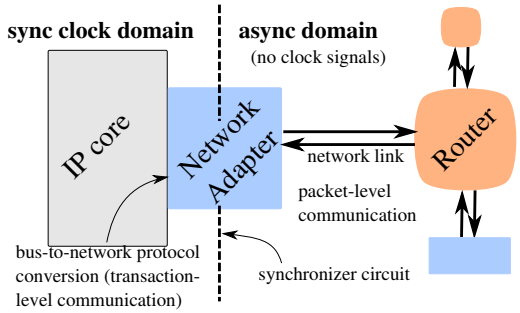


Figure 1: Component diagram of an asynchronous NoC.

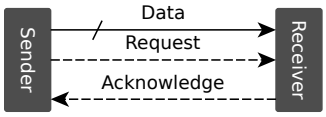


Figure 2: Signals of an asynchronous channel handshake.

An asynchronous channel, which transfers data from a sender to a receiver, is shown in Figure 2. Instead of a synchronous clock signal determining when data should be read, a handshaking protocol performs this function. Typically a *request* and *acknowledge* signal are used to accomplish this. The sender generates the request signal to indicate new data is available. The receiver responds with the acknowledge signal, indicating data is stored.

There have been a number of NoC proposals for incorporating storage and/or control logic within inter-router links. The work of *iDEAL* [4] showed that the performance penalty of reduced-complexity routers with few input/output buffers can be improved by using links containing storage elements. For a traditional synchronous NoC and mesh topology, moving storage to the links significantly reduced network power at a very slight performance reduction. Link pipelining is described for the *Xpipes* network components [5]. These are placed primarily to meet clock timing requirements. Error detecting link pipeline circuits were designed to achieve greater NoC robustness while maintaining high throughput [6]. Elastic buffers, similar to the asynchronous buffers used here, were used to reduce router complexity by using the link as a distributed FIFO buffer [7]. Throughput per energy was improved compared to the baseline architecture. Elastic and asynchronous link pipelining was explored in [8], but with an ad hoc approach in determining where and when a buffer should be inserted on a link. It also did not evaluate effects on large-message latency. A number of energy-efficient proposals, including pipelined links shared between multiple sources, is given by [9]. It uses a standard mesh topology and homogeneous SoC for evaluation and does not address the optimization problem of determining the the number of buffers on each link. Link pipelining for a delay-insensitive asynchronous NoC are described in [10], where multiple virtual channels can overlap packet transmissions at the flit level to maintain high link utilization. The paper did not describe the conditions or depth of the pipeline, nor was a system-level evaluation of the proposal given.

NoC optimization for a particular SoC is a rich topic to draw from. The COSI framework generates an application-specific NoC and floorplan, taking as input such constraints as core areas and average bandwidth between cores [11]. A heuristic search determines the topology and router configuration [12]. It uses floorplan information, router energy models, and core communication requirements. The results indicate a significantly reduced power and hop-count versus the best mesh topologies for a variety of SoC designs. For the QNoC routers, application-specific optimization is discussed in [13], but it focuses on mapping logical resources of a mesh-style topology rather than physical concerns. It presents a link capacity allocation method, but does not explain mechanisms for implementing such a method.

A large variety of async networks have been developed, but these have not provided details of link pipelining, even when it assumed. Fulcrum Microsystems created a large asynchronous crossbar to interconnect cores of a SoC [14]. The commercial startup Silistix, based on earlier academic research, sells EDA software and circuits that provide a customized asynchronous NoC, but has no published methods for the optimization process [15]. The MANGO router [16] provides both best-effort and guaranteed-service traffic. Faust [17] is a platform and fabricated chip used in 4G telephony development, and uses an asynchronous mesh-based NoC [18]. The QNoC group has developed an asynchronous router that provides multiple service levels and dynamically allocated virtual channels per level [19].

2. LINK PIPELINING

The goal of pipelining a link is to reduce the cycle-time, created by wire delay, by inserting a buffer between sender and receiver. Throughput along a link is increased, at the expense of single-flit latency and a slight power increase over only wire repeaters. This organization is illustrated in Figure 3, where a buffer is placed between a router and network adapter. Our asynchronous pipeline buffer is composed of a bank of latches (rather than flip flops) and a handshake controller. This arrangement is called a *linear controller* or *link buffer*. The use of latches saves almost half the area, and potentially power, compared to a traditional synchronous flip-flop design. Note, the pipeline buffer does not negate the need for wire repeaters (large inverters). In this work, we assume the buffers for the separate input and output channels are located in the same proximity, but this is not required. From this point forward, when we use the term *buffer* in the context of describing the network configuration we refer to the collection of latches and controllers for both channel directions.

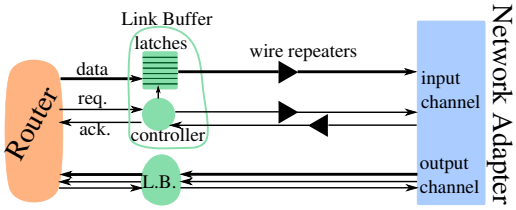


Figure 3: Organization of NoC link components showing detail for a router's output channel, and the equivalent link buffer (L.B.) for its input channel.

Our asynchronous router is designed for efficiency and simplicity, similar to other recent work [20]. Each switch directs a flit to one of two output ports. With bi-directional channels, this results in a three-ported router. Possible topologies include a ring, tree, or an irregular arrangement, but this work considers trees, as they have the minimal number of routers. The packet format consists of a single flit containing source-routing bits in parallel, on separate wires, with the data bits. We assume a transaction layer protocol implemented in the network adapter will use the packet's data field to transmit control information, such as addresses and command-type. This is not fundamentally different to packet formats that serialize a fixed packet size into multiple flits.

The packet is switched through a demultiplexer controlled by the most-significant routing bit. The bits are simply rotated for the output packet. The number of required routing bits is determined by the maximum hop count of a network, and this study required eight and twelve bits for the two benchmark designs. This format has the overhead of requiring routing bits with every flit, but does not require an extra header-flit carrying routing information common to other packet formats. Its energy overhead is further reduced in that series of packets following the same source-to-destination path do not cause switching in the routing bit wires.

This design uses an Artisan cell library on the IBM 65 nm *10sf* process. We used post-layout back-annotation to evaluate the router/link buffer's latency, HSPICE simulation for energy measurement, and leakage power from SOC Encounter. Latency and energy results are shown in Table 1, and assume 25% of bits switch each flit. Forward latency is the time between when an input channel receives a request signal to the time the router/buffer asserts the outgoing channel's request signal. Throughput is the maximum cycle-time of the component, assuming zero wire delay.

Table 1: Circuit-level properties of router and link buffer.

	Energy/flit (pJ)	Leakage (μW)	Latency (ps)	Throughput Gflits/sec
Router	1.03	9.76	460	2.1
Link Buf.	0.45	1.21	130	4.1

Insertion Strategies

We propose two strategies to determine under what conditions a link pipeline buffer should be inserted. The first, *path-specific* buffer insertion, pipelines links that require a throughput greater than a fraction, k, of the link's available bandwidth (ABW). The intuition behind this is that high-traffic links will benefit from additional buffering to reduce contention in the preceding routers, and also to decrease latency from a receiving router's *ack* signal (indicating the next flit may be sent) to a sending buffer *req* signal (indicating the next flit is ready).

This strategy is termed *path-specific* because the required throughput ($\mathcal{T}_{req}$) is derived from the source-to-destination paths that utilize the link; it is the sum of average throughput of each of these paths ($\mathcal{T}_p$). The number of buffers, $\aleph$, to insert on link ℓ is

$$\aleph_\ell = \left\lfloor \frac{\mathcal{T}_{req}}{(ABW_\ell \times k)} \right\rfloor \text{ where } \mathcal{T}_{req} = \sum_{path\ p\ using\ \ell} \mathcal{T}_p$$

The value of k is a user-parameter, and is explored in Section 4. In other words, this is an insertion threshold based on the percentage of link utilization.

For the application-specific SoCs we consider in this work, a common representation of core-to-core traffic requirements is a *core communication graph* (CCG) as shown in Figure 4. The expected average throughput required between cores is shown with an annotated edge. For this particular CCG, traffic is assumed to be equal in both directions, but a CCG is often shown with directed edges. Given a topology and CCG, each link in the topology is attributed a required throughput. Available link bandwidth (ABW) is based on link wirelength, as shorter links have faster cycle-time.

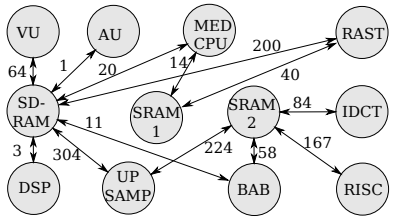


Figure 4: MPEG4 decoder *core communication graph*, with edges in MBytes/second.

The second strategy adds pipeline buffers to links with an available bandwidth (ABW) less than the throughput of the network adapters. We call these *core-throughput matching buffers* (CTMBs). For example, if a network adapter had a maximum throughput of 2 Gflits/s (yielding 64 Gbits/s with 32-bit flits), and a long link had a handshake delay of 700 ps (yielding 46 Gbits/s), the link would need one CTMB. This is analogous to wire pipelining for clocked networks when a link fails to meet the timing requirement derived from the clock period. For async systems, however, this is optional; links can be slower than the sending or receiving component and don't have to meet the "clock period." The intuition driving this strategy is to make sure cores are never slowed down by a wire delay in sending or receiving a flit. Even paths with low average bandwidth will send a series of flits one after another, and benefit from the increased link bandwidth. This advantage may or may not be worth the additional power overhead of the buffers, depending on system requirements and communication properties.

We integrated these two strategies within a tool that generates the network topology and placement, called *ANetGen* [21]. This tool searches for a network solution with the best power and latency characteristics, and generates a topology, floorplan, and SystemC simulator for the network.

3. METHODOLOGY

We evaluate our link pipelining proposals using two abstractions of SoC designs. One is an MPEG4 decoder SoC, originally described by [22] but with throughput changed to that shown in Figure 4. This benchmark has been used in other NoC research projects [12, 11]. The other is based on data given by Texas Instruments (TI-SoC), that in contrast to the MPEG4, has many more IP blocks (35) and communication paths (423 source-to-destination). Circuit properties for the MPEG4 design are assumed to be at the 65 nm technology node, with an area of 78.7 mm^2. The TI-SoC design assumes 32 nm parameters, thus the latency and energy of the routers and wires was adjusted accordingly.

The network adapter in the MPEG4 design had a throughput of 2 Gflits/s with 32 bits/flit, and for the TI-SoC, it had a throughput of 4 Gflits/s with 64 bits/flit.

Our network optimization tool, *ANetGen*, was configured to produce a topology and placement of the routers and pipeline buffers for various values of the path-specific k parameter. This was done for two sets of link pipeline configurations; with path-specific buffers only, and with both path-specific and CTMBs. The tool already optimizes link bandwidth on high-traffic paths by reducing their wirelength. This process already takes the "low hanging fruit" of bandwidth improvement; other experiments have shown that an asynchronous network with customized link bandwidths yield 46% less aggregate packet latency than a similar synchronous network with uniform link bandwidth [8]. Therefore, the results shown here are the *additional* improvements.

Simulations were run for each of the sets, where the k-parameter was varied to change the threshold of where path-specific buffers were inserted. For the MPEG4 set, k values were 1.0, 0.05, 0.03, and 0.02 resulting in the number of path-specific buffers of, 0, 1, 3, and 7, respectively. Due to the details of this SoC's specific traffic requirement, and the floorplan, there was only one link where path-specific buffers were inserted – the link connecting the *sram2* IP core to a router. When CTMBs are added, seven more buffers in total are inserted on the longest links. For the TI-SoC, k values of 1.0, 0.10, 0.05, 0.03 resulted in path-specific buffer counts of 0, 4, 18, and 42 respectively, spread throughout all the links. A total of 25 CMTBs are added, but some of these fall on the same links that have path-specific buffers, resulting in total link buffer counts (including both types) of 25, 27, 33, and 56. These k-threshold values were chosen experimentally.

Our simulator uses the Orion 2.0 wire models [23] to estimate dynamic wire energy, but we changed the wire leakage parameters to match the IBM process in our router and buffer evaluation. The Orion NVT library estimated this power to be between 5x and 7x greater than the 65 nm IBM process. In the TI-SoC 32 nm experiments, we scaled our router and link buffer energy by a factor derived from the Orion 2.0 router model energy at each node. The 32 nm circuit latency was reduced by half, and the network adapter throughput doubled. For all experiments we assume the fraction of data bits that switch in successive flits is 0.25.

The simulator uses a self-similar traffic generator, rather than the more common Poisson-distribution. This is based on the bursty traffic model [24], and is suggested as a key feature in future NoC benchmark sets [25]. A known volume of traffic to be sent during a known simulation duration is divided into two parts, each part weighted by the bias, b. Each of these is then split, and the process continues for each sub-division of time, until the desired time resolution is reached. Steps of the this process are shown in Figure 5.

This study uses *message latency*, rather than packet latency, because it seems to be a better metric to represent transfer of a useful piece of data, especially in media processing. The traffic generator outputs chunks of data, or *messages*, that simulate the requests from software to the transaction layer of SoC communication. The self-similar behavior is seen in the times of generation of these messages. A message is broken down into contiguous packets to be sent over the network as fast as it will allow. Message generation cannot be stalled by the network, but message latency through the network is actually a measure of network

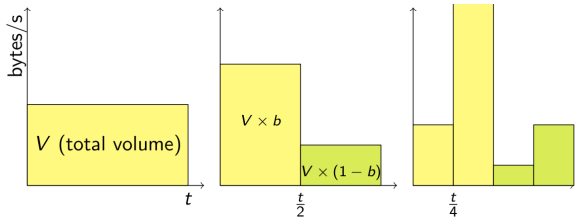


Figure 5: Generation of a self-similar traffic volume distribution over a simulation's duration.

bandwidth that considers the dynamic effects of packet contention with other paths. As such, message latency is defined as from when the first packet of the message leaves the sending core's output buffer and enters the network to the time the tail packet leaves the network and enters the destination core. This operation flow and structural partitioning is shown in Figure 6. For the MPEG4, we set the bias b-value to be 0.7 on each CCG path, a simulated time of 34 ms, and a 256-byte message size. The TI-SoC used a b-value of 0.5 and simulation duration of 8.3 ms.

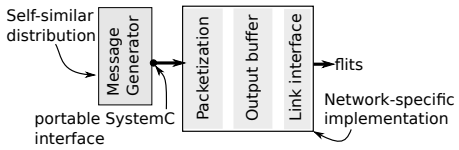


Figure 6: Structural partitioning of traffic generation.

Another performance metric we measured is the packet delay in a network-adapter's output buffer. This is defined as from the moment the traffic generator places a packet in the output buffer to the time the packet enters the network. The entry time is set for each packet by the traffic generator when it pushes an entire message to the buffer at once. Therefore, the last packet of a 64-packet message would have a minimum delay of 64 sender-cycles. The traffic generator operates detached from the network flow control so an infinite buffer is needed to accept its traffic at any time. The network then empties that buffer as quickly as possible. More details on the traffic generator, simulator, and models are described in [21].

The MEPG4 floorplan with 3 path-specific buffers on the *sram2* link and CTMBs is shown in Figure 7. Cores are labeled with their names, routers are red circles, and pipeline buffers are green squares. Logical connectivity between components is shown with black lines. Note this does not show actual "Manhattan" wire routing distances. Network adapters are assumed to be placed where a link is attached to a core. Notice that buffers are equally spaced across a link. The longest, from *au* to router *4* has three pipeline buffers, pb0, pb1 and pb2. The path-specific buffers connecting *sram2* are pb7,8,9.

4. RESULTS

Power and Latency

A succinct metric to determine the benefit of adding link pipeline buffers is *power-latency product* (PLP) of the network. This is similar to the *energy-delay product* commonly used in CPU architecture or VLSI comparisons. The *power*

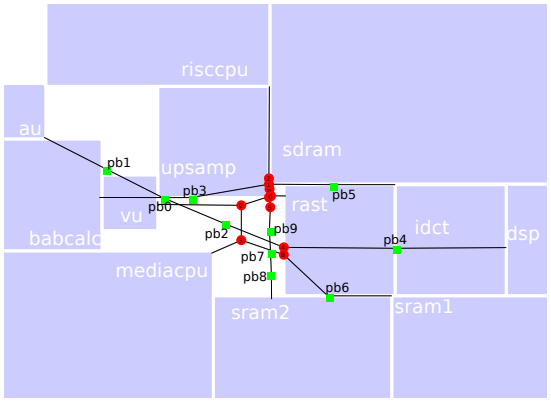


Figure 7: SoC floorplan showing cores, routers, buffers, and topology, cropped and edited for legibility.

term is the sum of dynamic and leakage power of the routers, wires, and wire pipeline buffers. The *delay* term is the mean packet latency through a network adapter's output buffer, added to the mean message (256 bytes) latency through the network. Delays were normalized to give equal weight to network and output buffer latencies. Figure 8 shows this metric on the Y-axis, with the X-axis showing the total number of buffers inserted on all links, including both CTMBs and path-specific. Data is given in two series, with and without additional core-throughput matching buffers.

For the MPEG4 benchmark, the addition of one path-specific buffer greatly lowers (improves) PLP. A slight additional improvement is seen with the addition of a few more path-specific buffers, but PLP gets worse after this. This trend is similar when CTMBs are also used, indicated by the darker line data. This chart is also useful in comparing the benefit of using the CTMBs. With just CTMBs and no path-specific buffers, the PLP is improved, but only very slightly from the initial solution. The best solution shown in the data is with three path-specific buffers as well as CTMBs (10 total network buffers). PLP worsens with more path-specific buffers (14 total buffers), but is better than if CTMBs were not used (7 total buffers).

For the TI-SoC design the results are quite different. This stems from the much larger number of IP blocks and paths, as well as a more uniform distribution of the communicating blocks. Generally, the addition of buffers helps greatly, but CTMBs are a more efficient strategy from the standpoint of mean latency over all paths. The addition of a low number of path-spec buffers only (totals of 10–20) reduces PLP a modest 20%. If the power budget can allow 30–50 total buffers, it is better to add CTMBs, which show a large reduction of over 50% in PLP.

Power consumption of various network configurations is shown in Table 2. For these experiments the topology and placement are left constant, with only the number of buffers varying. Therefore, the power of the routers and wires is constant because the same traffic is sent in each trial and the total wirelength is the same. The dynamic and leakage power of routers and wires, common to all configurations, is shown at the top of the table. Configurations are organized in sets, separated by benchmark (MPEG4 and TI-SoC) and by buffer insertion type (only path-specific buffers and both path-specific and CTM buffers). A lower value of k-threshold represents an increased number of path-specific

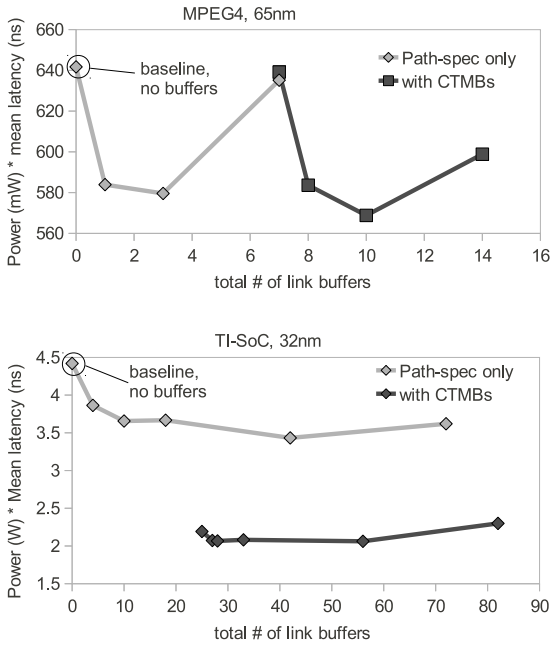


Figure 8: Power-latency product for various numbers of buffers inserted based on link utilization percentage. Results shown with and without core-throughput matching buffers.

buffers, with a value of 1 having none. For each of these, the buffer power (sum of dynamic and leakage) is shown, along with the total network power. For clarification, the wire power is due to the drivers/repeaters (large inverters) needed along a link's length. The much greater power consumption of the TI-SoC is due to it sending far more aggregate traffic. The total power consumption rises slightly in both benchmarks with the addition of more link buffers, as expected. Link buffer power is a small portion of the total, less than 10% in most configurations.

Table 2: Power (mW) of various buffer configurations.

Power values common to all configs				
	Rtrs dyn	Rtrs leak	Wire dyn	Wire leak
MPEG4	1.34	0.29	3.28	0.84
TI-SoC	47.8	1.63	34.6	16.70
	path-specific k threshold			
MPEG4	1	0.5	0.3	0.2
Without CTMBs				
Link Buffers	0	0.12	0.36	0.85
Total	5.76	5.88	6.13	6.61
With CTMBs				
Link Buffers	0.06	0.18	0.42	0.90
Total	5.82	5.94	6.18	6.66
TI-SoC	1	0.1	0.5	0.3
Without CTMBs				
Link Buffers	0	2.61	9.01	20.72
Total	100.7	103.3	109.7	121.5
With CTMBs				
Link Buffers	3.37	5.27	9.78	21.67
Total	104.1	106.0	110.5	122.4

Mean latencies are shown in Table 3 for packets through the network adapter's output buffer for messages through the network. Both of these metrics generally improve when

Table 3: Mean latency (ns) of buffer configurations.

Measurement Location	path-specific k threshold			
MPEG4	1	0.5	0.3	0.2
Without CTMBs				
Core's Output Buffer	15281	12257	11430	11644
Network	55.74	54.58	52.92	53.58
With CTMBs				
Core's Output Buffer	15332	12400	11248	10693
Network	54.92	53.78	51.72	51.59
TI-SoC	1	0.1	0.5	0.3
Without CTMBs				
Core's Output Buffer	11.8	9.7	8.4	6.5
Network	21.9	19.4	17.8	16.2
With CTMBs				
Core's Output Buffer	10.1	8.6	8.2	6.2
Network	19.1	17.9	17.3	15.7

more path-specific buffers are used. The large output buffer latency in the MPEG4 benchmark is an effect of lack of backpressure to the traffic generator; it is a useful *relative* comparison, but does not reflect latencies expected in an actual system. The network message latency is more representative of actual system behavior.

Message Latency per Path

Besides overall mean message latency, another picture of performance is seen by looking at the message latencies for each source-to-destination path in the SoC. For example, the MPEG4 path from source core *sram2* to destination core *risccpu* has a median latency of 65 ns with no link buffering, as seen in Figure 9 with the path-ID `21`. These measurements are from the time a 256-byte message has begun its exit of a core until it completely enters a receiving core. This is a measure of available bandwidth on a path, while considering dynamic effects of contention with other paths.

The addition of link buffering improves median latency significantly on some paths, notably `20,21,22` which carry high traffic from *sram2*. Latency rises slightly with the addition of more buffers. Buffers were added to the link connecting the *sram2* core to a router, which explains the benefit on those paths. Other paths do not benefit from these added buffers. The effects on maximum message latency is not conclusive, although a few paths seem to benefit slightly, such as `0` (*au*→*sdram*) and `18` (*sram1*→*rast*). This is from reduced contention, a side-effect of the improved connection to *sram2*.

The effects on per-path message latency by adding CTMBs is shown in Figure 10, for the MPEG4 benchmark. All values on the chart are normalized to the configuration with only path-specific buffers (no CTMBs) inserted with the same k-threshold. The data series represent various k-thresholds and thus different numbers of path-specific buffers. Paths that showed little difference with the addition of CTMBs were removed from the figure. Median latency is improved on many paths, and is an indication of increased throughput when the network is uncongested. Note that the paths improved with CTMBs are different than those improved with path-specific buffers; paths `20,21,22` did not show much change. Also interesting is that even though many paths have a median latency reduction, the mean latency considering all paths (shown in Table 3) was not improved much with the addition of CTMBs. This is due to the fact that the paths carrying the greatest traffic already have buffers assigned to them as the *path-specific* type. The other paths do see an improvement, but it does not greatly impact mean network latency because they carry less traffic.

Maximum latency improvements were mixed, with some drastically worse paths, and many slightly improved ones. The paths showing worse maximum latency are the topologically longest, and thus have the highest probability for contention and delay. The addition of CTMBs increases the rate messages can enter the network, but not necessarily provide beneficial throughput increases "downstream." The effect is, in the worst case, longer waiting times within the network rather than in the core's output buffer. The benefit of path-specific buffers to maximum latency seems to apply less broadly than median delay benefit. However, some paths do benefit from an increasing number of buffers such as `path 21` in 65 nm, connecting *sram2*→*risccpu*.

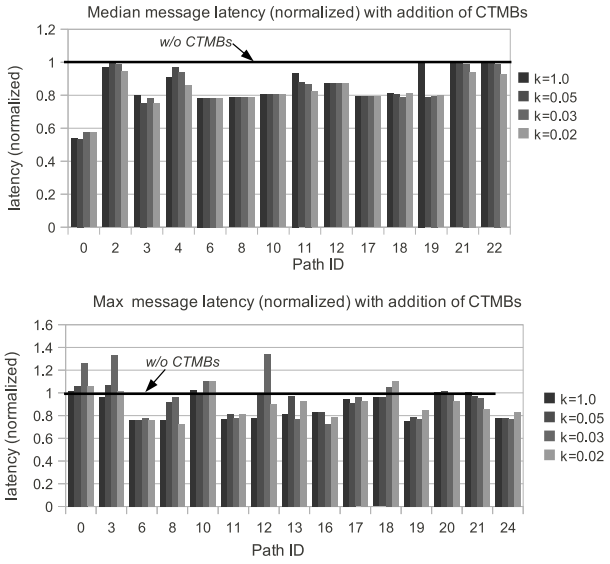


Figure 10: Change in message latency through the network with the addition of CTMBs, for the MPEG4 benchmark. Latency is normalized to the network configuration with path-specific buffers only. Only paths exhibiting change are shown.

The per-path message latency evaluation for the TI-SoC benchmark is done in a different way than for the MPEG4 because it has many more paths. Instead, we show a histogram of the number of paths that improve, for various buffer configurations, compared to the baseline network with no buffers. Figure 11 shows this histogram for two sets: path-specific buffers only and both path-specific and CTMBs. The better a network performs, the more paths it will have at lower latencies, with a value of 1 being equal to the baseline. A larger number of path-specific buffers reduces latency on more paths, both without and with CTMBs. A few paths had worse latency by 5-10% when many buffers are used, but many more paths showed improvement by 10-50% less latency. The benefit of CTMBs alone without path-specific buffers is seen in 11b with the 1.0 k-threshold series. The additional series show the added CTMBs.

These results show that the addition of link pipeline buffers can improve performance as indicated by message latency at the cost of a small power increase. The optimal insertion parameters are design-specific, and the space should be

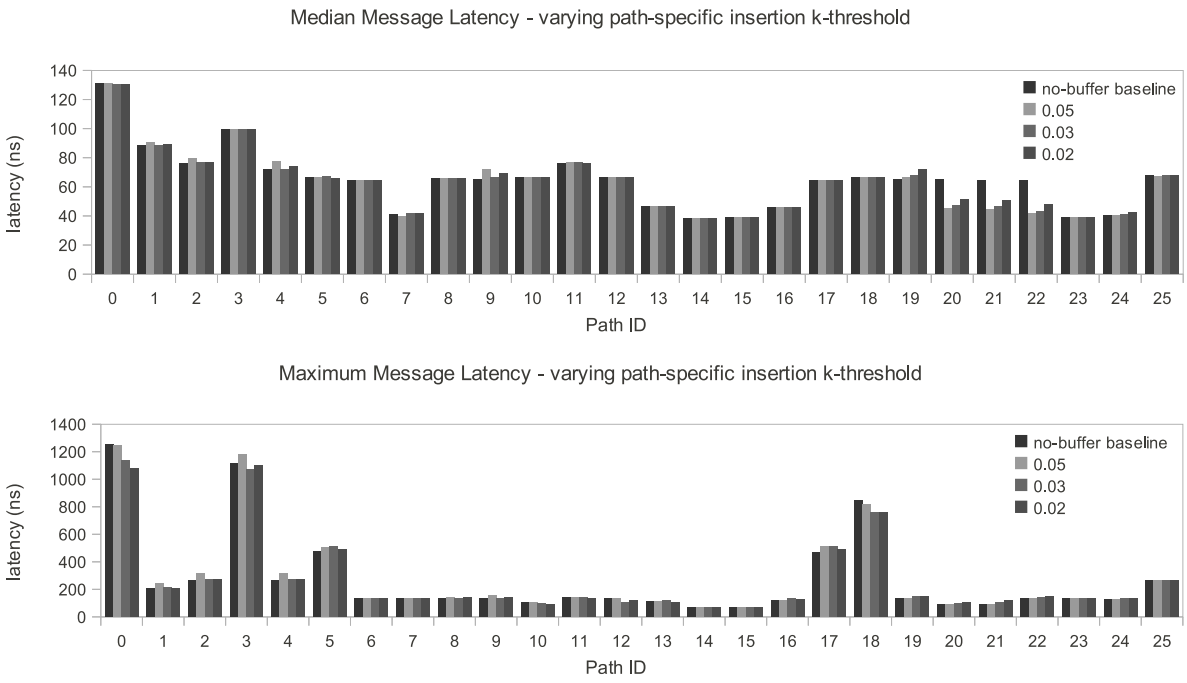


Figure 9: Median and maximum message latencies for the MPEG4 network for each source-to-destination path. Data series represent different numbers of path-specific buffers (more buffers for a lower k-threshold) and no CTMBs.

explored by the designer. In the MPEG4 design, the most benefit came from path-specific buffers, while the TI-SoC design was most sensitive to CTMBs. Path-specific buffers yield improvements on a few critical paths, and seem to be best when the addition of the fewest number of buffers is desirable. The use of CTMBs offer decreased median and maximum message latency for many paths, at the expense of increased maximum latency on some. The benefit is more widespread with CTMBs, and the most efficient network for each design used CTMBs and a moderate number of path-specific buffers.

5. CONCLUSION

In this work we have introduced two strategies for determining which links of an asynchronous NoC should be pipelined, and incorporated them into our NoC optimization tool. An advantage of an async network is that this optimization can be targeted at specific links in the network, guided by expected traffic patterns. This is in contrast to a classic synchronous NoC (not including source-synchronous methods) that change frequency on the whole network, or use additional synchronizers between clock domains of different frequencies.

For two benchmark SoCs, we varied the number of link pipeline buffers using combinations of the two strategies. Simulation results showed the greatest benefit to power*latency product was using CTMBs along with some number of path-specific buffers. The path-specific buffer insertion strategy is *complexity-effective* relating to energy, in that it increases performance more than it costs in energy, with and without the use of CTMBs.

The NoC-engineer should take these results into consideration as evidence that the link pipelining design space should be explored for the configuration appropriate for the given requirements. A SoC design may be more sensitive to one insertion strategy or the other. Generally, a design with a low number of critical, high throughput paths will see the most gain from path-specific buffers, while a design with a large number of lower throughput paths will benefit most from CTMBs. The given strategies offer the NoC engineer another knob to turn for setting the power–performance point.

Acknowledgments

This research is supported by the U.S. National Science Foundation under grant CCF-0810408, CCF-0702539, and Semiconductor Research Corporation under task 1817.001. We would like to thank ARM and IBM for providing the 65 nm library cells and process technology.

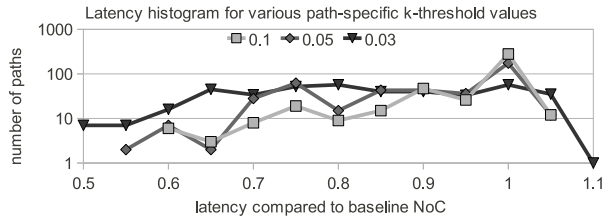
6. REFERENCES

[1] I. M. Panades, F. Clermidy, P. Vivet, and A. Greiner, "Physical implementation of the dspin network-on-chip in the faust architecture," in *Proc. Int'l Symp. on Networks-on-Chips*, 2008, pp. 139–148.

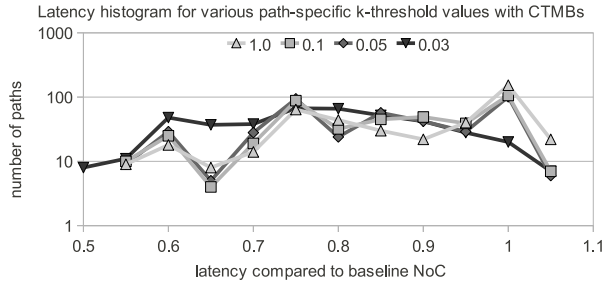
[2] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[3] T. Bjerregaard, S. Mahadevan, R. G. Olsen, and J. Sparsø, "An OCP compliant network adapter for GALS-based SoC design using the MANGO network-on-chip." in *Proc. of Int'l Symp. on System-on-Chip*, 2005.

[4] A. K. Kodi, A. Sarathy, and A. Louri, "ideal: Inter-router dual-function energy and area-efficient links for network-on-chip (noc) architectures," in *Proc. Int'l Symp. on Computer Architecture*, 2008, pp. 241–250.



(a) insertion of various numbers of path-specific buffers, and no CTMBs



(b) CTMBs added, with increasing numbers of path-specific buffers

Figure 11: Histograms showing the number of paths with various median message latencies for the TI-SoC design. Latencies are normalized to the baseline configuration with no link buffers. Improvement is indicated by more paths of less latency, which occurs as the k-threshold is decreased and thus more path-specific buffers are added.

[5] D. Atienza, F. Angiolini, S. Murali, A. Pullini, L. Benini, and G. De Micheli, "Network-On-Chip Design and Synthesis Outlook," *Integration-The VLSI Journal*, vol. 41, no. 2, Feb. 2008.

[6] R. R. Tamhankar, S. Murali, and G. De Micheli, "Performance driven reliable link design for networks on chips," in *Proc. Asia and South Pacific Design Automation Conference*, 2005, pp. 749–754.

[7] G. Michelogiannakis, J. Balfour, and W. Dally, "Elastic-buffer flow control for on-chip networks," in *Proc. Int'l Symp. on High Performance Computer Architecture*, 2009, pp. 151 –162.

[8] J. You, D. Gebhardt, and K. S. Stevens, "Bandwidth Optimization in Asynchronous NoCs by Customizing Link Wire Length," in *International Conference on Computer Design*. IEEE, Oct 2010, pp. 455–461.

[9] J. Kim, "Low-cost router microarchitecture for on-chip networks," in *Proc. Int'l Symp. on Microarchitecture*, 2009, pp. 255–266.

[10] T. Bjerregaard and J. Sparso, "Implementation of guaranteed services in the mango clockless network-on-chip," *Computers and Digital Techniques*, vol. 153, no. 4, pp. 217 – 229, 2006.

[11] A. Pinto, L. P. Carloni, and A. L. S. Vincentelli, "A methodology for constraint-driven synthesis of on-chip communications," *IEEE Transactions on Computer Aided Design*, vol. 28, no. 3, pp. 364–377, 2009.

[12] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Proc. Int'l Conf. on Computer-Aided Design*, 2006, pp. 355–362.

[13] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Network delays and link capacities in application-specific wormhole nocs," *VLSI Design*, vol. 2007, 2007.

[14] A. Lines, "Asynchronous interconnect for synchronous soc design," *IEEE Micro*, vol. 24, no. 1, pp. 32–41, 2004.

[15] W. J. Bainbridge and S. B. Furber, "CHAIN: A Delay Insensitive CHip Area INterconnect," *IEEE Micro special issue on Design and Test of System on Chip*, vol. 142, No.4., pp. 16–23, Sept. 2002.

[16] T. Bjerregaard and J. Sparsø, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proc. Design, Automation, and Test in Europe*, 2005, pp. 1226–1231.

[17] D. Lattard, E. Beigne, C. Bernard, C. Bour, F. Clermidy, Y. Durand, J. Durupt, D. Varreau, P. Vivet, P. Penard, A. Bouttier, and F. Berens, "A telecom baseband circuit based on an asynchronous network-on-chip," *Solid-State Circuits Conference, Digest of Technical Papers*, Feb. 2007.

[18] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin, "An asynchronous noc architecture providing low latency service and its multi-level design framework," in *Proc. Int'l Symposium on Asynchronous Circuits and Systems*, 2005, pp. 54–63.

[19] R. R. Dobkin, R. Ginosar, and A. Kolodny, "Qnoc asynchronous router," *Integr. VLSI J.*, vol. 42, no. 2, pp. 103–115, 2009.

[20] M. N. Horak, S. M. Nowick, M. Carlberg, and U. Vishkin, "A low-overhead asynchronous interconnection network for gals chip multiprocessors," in *Proc. Int'l Symp. on Networks-on-Chips*, 2010, pp. 43–50.

[21] D. Gebhardt, J. You, and K. S. Stevens, "Comparing energy and latency of asynchronous and synchronous nocs for embedded socs," *Proc. Int'l Symp. on Networks-on-Chips*.

[22] E. B. V. D. Tol and E. G. T. Jaspers, "Mapping of mpeg-4 decoding on a flexible architecture platform," in *Media Processors*, 2002, pp. 1–13.

[23] A. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *DATE*, April 2009, pp. 423–428.

[24] M. Wang, T. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos, "Data mining meets performance evaluation: fast algorithms for modeling bursty traffic," in *Proc. International Conference on Data Engineering*, 2002.

[25] Z. Lu, A. Jantsch, E. Salminen, and C. Grecu, "Network-on-chip benchmarking specification part 2: Micro-benchmark specification," *Technical Report, OCP International Partnership Association, Inc.*, 2008.