A Tree Arbiter Cell for High Speed Resource Sharing in Asynchronous Environments

Syed Rameez Naqvi and Andreas Steininger
Department of Computer Engineering, Vienna University of Technology, Austria
{rnaqvi, steininger}@ecs.tuwien.ac.at

***Abstract*—We present a novel tree arbiter cell that allows a pipelined processing of asynchronous requests. In this way it can achieve significantly lower delay in the critical case of frequent requests coming from different clients. We elaborate the necessary extension to facilitate a cascaded use of this cell in a tree-like fashion, and we show by theoretical analysis that in this configuration our cell provides better fairness than the standard approach. We implement our approach and quantitatively compare its performance properties with related work in a gate-level simulation. In our sample asynchronous Networks-on-Chip application our new cell proves to increase the throughput of three different designs available in literature by approximately 61.28%, 69.24%, and 186.85% respectively.**

I. Introduction

Networks-on-Chip (NoCs) have become the de-facto standard for interconnect within Systems-on-Chip (SoCs). They comprise a grid of routers with physical links between them, and communication within the SoC is established by forming an appropriate path of these physical links. Although in detail many different switching schemes are in use, they all have in common that the physical links are *shared* as they can be potentially used for various communication paths. Accesses to the physical link need to be appropriately orchestrated among the potential clients to avoid collisions. While this is relatively easy to achieve in a globally synchronous system, the modern Globally Asynchronous, Locally Synchronous (GALS) SoCs with their multiple, often uncorrelated, timing domains usually employ a much more sophisticated arbitration. One solution here is to synchronize all clients' requests to one time domain, thus reducing the problem to the globally synchronous case. However, synchronizers exhibit imperfect reliability, and they incur a performance penalty [1]. The other solution is to employ dedicated arbitration circuits (or just *arbiters*), the genuine purpose of which is to resolve conflicting requests, and to allow their arrival times to be arbitrary. Since the asynchronous NoCs make heavy use of arbitration, any delay introduced by the arbiter will degrade the NoC performance in terms of latency and throughput. With this motivation we propose a novel arbiter circuit in this paper that is faster than existing ones, as it allows pipelined, rather than strictly sequential, processing of accesses in an asynchronous setup.

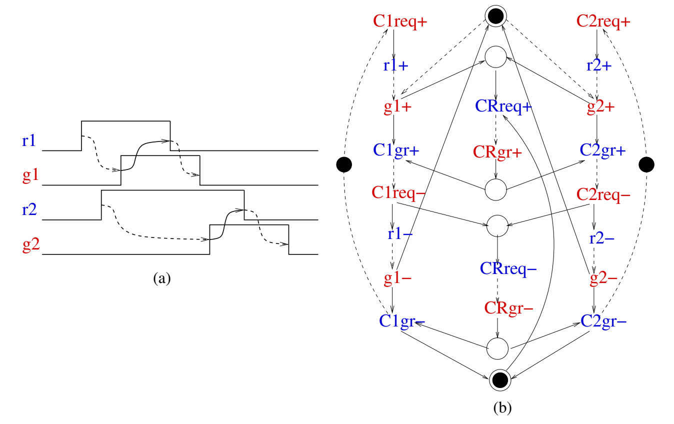

Fig. 1. STG of a 4-phase Two Input TAC

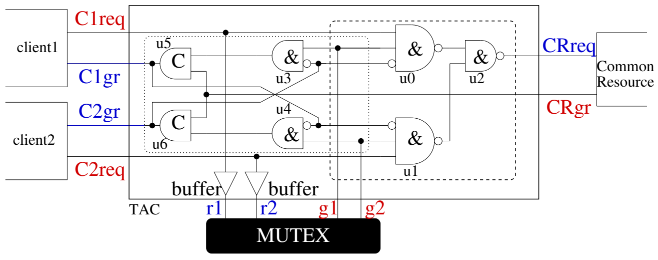

Fig. 2. Gate level netlist of a 4-phase Two Input TAC

II. Background and Related Work

The purpose of an arbiter is to control the access of several clients to a shared resource (like a physical link in a NoC). It is known that if the contending requests arrive at (nearly) the same time, choosing the winner may involve the risk of metastability. A mutual exclusion (MUTEX) circuit [1] is therefore employed for this decision. Its operation using a 4-phase protocol is depicted in fig. 1(a): The activation of the earliest request $r1$ is acknowledged by a grant $g1$ that remains activated until $r1$ is withdrawn. At that point the second request $r2$, if activated, will get serviced similarly.

Although in principle a multi-way MUTEX can be built [1], the simpler two-way MUTEX is often preferred, and the remaining arbitration logic is arranged in several levels, so as to extend a two-way (n-way) arbiter to a four-way (2n-way) arbiter. The circuit used for this purpose is the *tree arbiter cell* (TAC); its low latency 4-phase version that serves two clients has been proposed by Yakovlev et al. [2], and we will use it throughout the rest of the paper to illustrate our approach.

Like most of the asynchronous circuits, the TAC is modeled using a signal transition graph (STG) presented in Fig. 1(b), which synthesizes to the circuit shown in fig. 2. An STG [1] is an interpreted petrinet, and it depicts the dependies between the input and output signals.

A close analysis of the TAC's STG reveals a possible window of improvement (in terms of its latency): On the one hand, the *down* transitions on clients' requests ($C1req-$, $C2req-$) release the MUTEX ($r1-$, $r2-$) and the common resource ($CRreq-$) simultaneously. On the other hand, however, the requests to lock the arbiter ($C1req+$, $C2req+$) first acquire the MUTEX ($r1+$, $r2+$), followed by reserving the common resource ($CRreq+$) in series. Ghiribaldi et al. [3] recently proposed an efficient 4-way arbiter that exploits the same window to achieve higher performance in their arbitration scheme. A MUTEX forms the common resource which is reserved and released in parallel with the local MUTEXs. This roughly saves delay of a MUTEX and that of a standard gate.

Some *fast* fixed priority programmable arbiters already exist in literature that work perfectly in synchronous environment (routers) [4], [5]. All those arbiters, in addition to the arbitration circuitry, require scheduling logic to dynamically generate new priority vectors every clock cycle. Although such priority vectors may be generated for asynchronous environments as well, the arbitrary arrival time of the input requests in that case makes the choice of the winner tedious, and usage of the MUTEX, in any case, becomes mandatory. Furthermore, the scheduling logic itself incurs area, power, and performance penalties. All these observations make such arbiters irrelevant to asynchronous design styles.

Felicijan proposed a low latency static priority asynchronous arbiter in [6]. The design did not need any explicit scheduling logic, since it comprised a linear priority module, which allowed one of the n clients, c_k, to block the rest $(k-1)$ having lower priority. The drawback associated with this approach is the number of MUTEXs that scales linearly with clients, and the number of Muller C-elements also scales badly.

In this work we propose an arbitration logic for fair and nondeterministic decision (where all clients have the same priority). Like [6] we assume that the arbiter is not the slowest component of the design, and that the clients, once received the grant, would keep the arbiter reserved at least for the duration longer than the latency of the arbiter. These assumptions are not so optimistic, especially for switch controllers in NoCs, where the header flit reserves the shared resources, and the tail flit releases them. Although our methodology is equally applicable to Ghiribaldi's approach [3], we only emphasize on the TAC because of its simple architecture that allows systematic modifications that are relatively easy to illustrate.

III. Proposed Tree Arbiter Cell

A. Window of Improvement

Refer to fig. 1(b). Once the common resource (further called CR) is acquired ($CRgr+$) and the grant to one of the clients is set (e.g., $C1gr+$), then the control waits for the corresponding client's request to go low ($C1req-$) before the MUTEX and the CR may be given to the other client (if already active). Now assume that $C1$ takes indefinitely long to remove its request, which forces $C2$ to wait indefinitely as well. Let's denote this *waiting time* ($C1gr+ \rightarrow C1req-$) as wt_{C1}. Once $C1$ has set its request low, a sequence of events must take place before $C2gr+$ could happen (We refer to the sequence $Cxreq- \rightarrow Cygr+$ as the *handoff* H_{CxCy} in the following). These include the release of the MUTEX and the CR (propagation delay of the arbiter while a grant switches from high to low, *tphl*), which happen simultaneously, followed by the events needed to reserve both of them for $C2$ (propagation delay when a grant switches from low to high, *tplh*). While it is not possible to optimize wt_{C1} (a client can keep the arbiter locked for as long as it needs it), the latter two, being local to the TAC, however, can be reduced. In fig. 2, the gates $\{u3, u5\}$, and $\{u4, u6\}$ form a mutual interlocking mechanism, through which $C1gr+$ prevents $C2gr+$ to happen, and vice versa. Consider if, in our example, *g2* would be already active, and the CR reserved when $C1gr-$ happens, then $C2gr+$ only sees a small delay, thereby eliminating *tphl*, and significantly optimizing *tplh*.

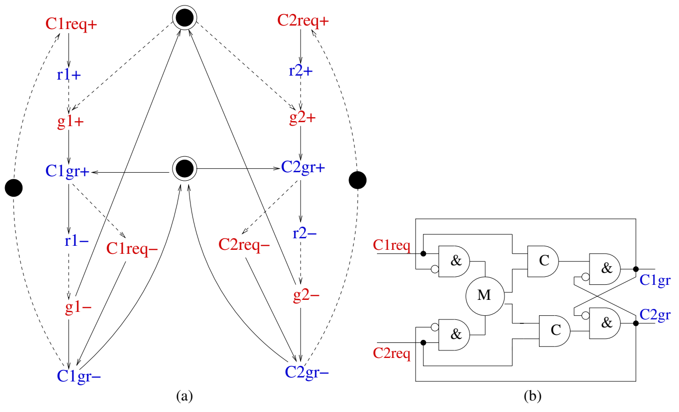

Fig. 3. Proposed 2-way Arbiter: (a) STG, (b) conceptual schematic

B. Design Concept

For simplicity we begin describing our methodology for the case of two clients. Fig. 3(a) presents the STG of our arbiter, with all the instances relevant to the CR removed. A small difference that we have brought into this mechanism is the release of the MUTEX ($r1-$, $r2-$) without waiting for the client's request going down: since the mutual interlocking already prevents the pre-mature handoff, the MUTEX is free to be allocated to the other client. In simple words, $Cygr+$ will not happen until $Cxreq-$ has happened, however *Cy* already possesses the MUTEX. Fig. 3(b) presents a possible implementation of the STG. Although our strategy allows the waiting client to acquire the MUTEX while the winner is yet to release it knowingly, one still might argue that the delay introduced through the C-gates and the cross coupled AND-gates, would be larger than that for releasing/acquiring the MUTEX itself. If so, there is no obvious benefit in terms of performance, and we unnecessarily introduced a number of gates in the system. The benefit, however, will become visible when we apply the same strategy to the TAC, discussed next.

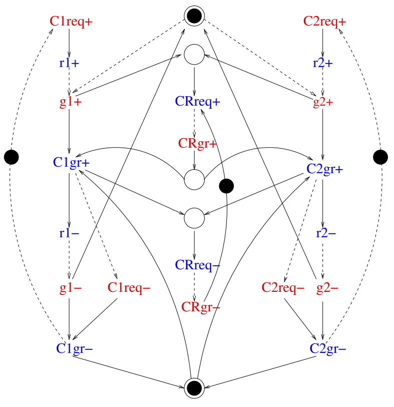

Fig. 4. STG of the proposed TAC

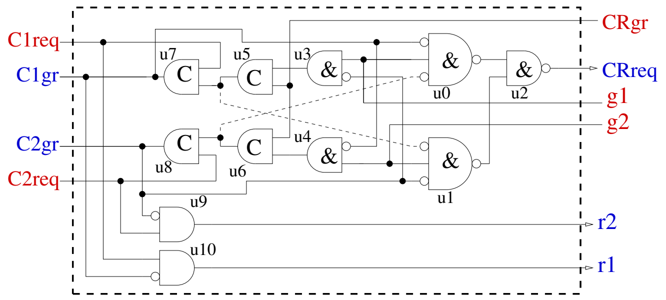

Fig. 5. Circuit of the proposed 2-way Arbiter

C. Adaptation to TAC

The STG shown in fig. 4 illustrates how the TAC adapts to our modifications. A client may acquire the arbiter by following the identical sequence of events as in case of the standard TAC, i.e., the requesting client first acquires the local MUTEX, followed by acquiring the common resource (like [3] we use a MUTEX here as well, further called CRM). The difference that we propose shows up in the other phase: As soon as one of the clients receives the grant, the CRM and the local MUTEX are simultaneously released, and the other client is allowed to lock both of them. The latter type of locking may be termed as *virtual locking*, since the first client is still in charge of the CRM until it literally lowers its request. This STG when synthesized generates the schematic shown in fig. 5. Apparently it seems to have incurred a further overhead in terms of area and latency; the benefit, however, is that *tphl* no more depends on wt_{Cx} (it is initiated as soon as the grant is given). So far we have simply generated a circuit that meets our criterion. However, there are some other challenges that we address in the following in turn.

1) Local clients' interlocking: As may be observed in fig. 5, the handoff H_{CxCy} still sees a delay of three C-gates and an AND-gate[1]. Since this interlocking is essential it cannot be avoided, but this latency may be reduced to two C-gates by adopting the interlocking shown in fig. 6. Note that all the C-gates used in the method are asymmetric, i.e., the inverted inputs are only relevant during *low to high* transitions of

[1]Recall that we assume wt_{Cx} is sufficiently large to allow the other client to acquire the local MUTEX and virtually reserve the CRM in the meantime.

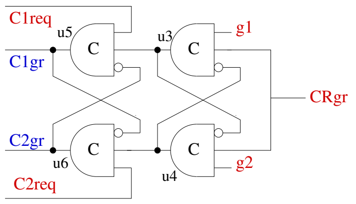

Fig. 6. Proposed rapid interlocking within 2-way Arbiter

the gates; symmetric C-gates in place could easily lead to deadlocks. While *u5* and *u6* interlock each other to prevent the premature handoffs, *u3* and *u4* do the same to guarantee a safe handshake protocol with the CRM.

A small improvement in terms of performance may be brought into the circuit by adopting the methodology proposed in [3]: The CRM may be reserved simultaneously with the local MUTEX. This only requires the gates *u0* and *u1* to have *C1req* and *C2req* as inputs, instead of *g1* and *g2* respectively.

2) Interlocking multiple TACs: In a tree structure, arbitrating multiple clients ($n = 2^k$) requires multiple TACs ($\sum_{i=1}^{k-1} 2^{k-i}$) to be employed in $log_2 n - 1$ levels, fig. 7. At the last level ($n = 4$), a CRM may be placed as the common resource for both the TACs, as already mentioned. Interestingly, as soon as one of the TACs, say *T1*, has released the CRM, while one of its grants, say *C1gr*, is still high, the other TAC, say *T2*, may acquire the CRM, and one of its grants, say *C3gr*, may also go high, thus violating the most fundamental property of an arbiter. This happens because our pipelined scheme relies on the mutual interlocking between the two grant outputs *within* a TAC, which is not effective across different TACs. Just like *C1* prevents *C2* within *T1* from getting a grant in parallel via interlocking, it must also do the same to *C3* and *C4* in *T2*. A speed independent solution will require complete handshaking between the two TACs, which may be inserted in the STG of fig. 4: The winning TAC, *T1*, sends out a *lock_others_request* signal right after $CRgr+$. On *T2* the same signal appears as *lock_me_request*, which must be mutually exclusive with $CRgr+$, since the CRM already belongs to *T1*. At this point *T2* sends the acknowledgment, and does not allow any of its grants to go high until the unlock signal has arrived from *T1*. This completes the handshake protocol between the contending TACs. The STG in fig. 8 depicts this logic (for simplicity we have shown the case of a single client), which guarantees speed independence. This solution, however, may incur a performance penalty since *T1* has to wait for the acknowledgment before its $Cxgr+$ could happen. This may be slightly relaxed by making the following timing assumptions.

3) Timing assumptions: We insert an AOI-gate between the output grants of the TAC that generates the *lock_others_request* signal, and a pair of AND-gates (one for blocking each client) that implements the required functionality of the *lock_me_request* signal, as shown in fig. 9. Recall from the previously discussed scenario that *T1* and *T2* can generate simultaneous grants if *lock_others_request* from the former does not reach the latter timely. The condition for safe interlocking is evaluated as follows: *T1* while releasing the

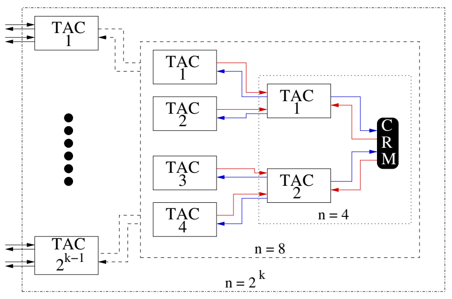

Fig. 7. Arrangement of TACs in several levels

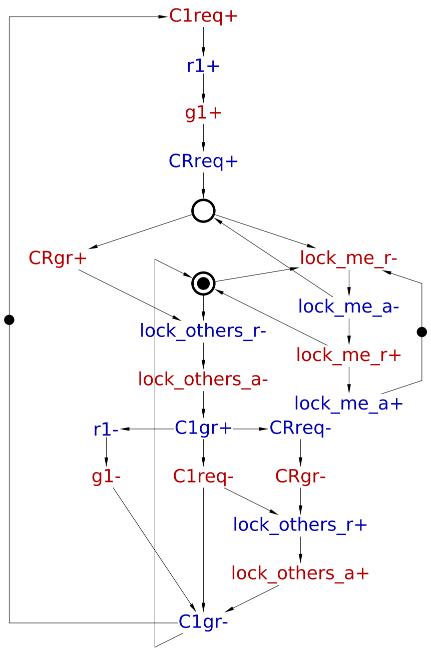

Fig. 8. STG of the proposed TAC with multiple TACs interlocking

CRM roughly sees the delay of two NAND-gates ($u0+$, $u2-$), handoff at the CR ($CRgr_{T1}-$, $CRgr_{T2}+$), and a wire delay[2] between the TAC and the CRM, refer to fig. 5. On the other hand, *T2* between reserving the CR and generating a grant sees two C-gates ($u5+ \rightarrow u7+$, or, $u6+ \rightarrow u8+$) and a wire delay. *T1* can block *T2* within a delay of one AOI-gate. For the TAC interlocking to be safe, the following condition must hold true, which we believe is quite simple to achieve;

$$\delta(AOI-) + \delta(wire) < \delta(u5+) + \delta_i(u7+) + \delta(wire)) + \delta(u0+) + \delta(u2-) + \delta(H_{CRM})$$

Here *δ(X)* and δ_i*(Y)* refer to the switching and inertial delays of gates *X* and *Y* respectively. H_{CRM} refers to the handoff at the CRM. For simplicity, if we assume similar wire delays on both sides of the equation, then the rest suggests that the AOI-gate must be faster than a C-gate, two NAND-gates and a MUTEX connected in series, which is very safe.

D. Unfairness Window

Fair arbitration demands that the grant be given to the first amongst all the clients that requested the CR. Neither the design of the TAC, nor of that proposed in [3] fulfils this demand. For example, consider a scenario where clients *C1* and *C2* request for the CRM simultaneously through *T1*, and *C1* wins the grant while *C2* is put to wait. Now after a while *C3* makes its request through *T2*, which will immediately reach the other input of the CRM, since *C4* is still inactive on the same TAC. Although *C2* made its request before *C3*, the latter would unfairly win the grant as soon as *C1* had released it. This unfairness happened because the request from *C2* to the CRM still had to go through two NAND-gates within *T1*, and this time was sufficient for *T2* to acquire the CRM. In fig. 10 we have presented the maximum length of this unfairness window, i.e., the period during which a client can unfairly reserve the CR. The abbreviations *CR* and *M* refer to the delays inserted due to the CRM and the local MUTEX respectively, and *FL* and *BL* refer to the forward and backward latencies of the TAC ($Cxreq+/- \rightarrow CRreq+/-$, $CRgr+/- \rightarrow Cxgr+/-$). Note that the events shown in part (a) of the figure correspond to the response of the standard TAC, and (b) presents the behavior of the proposed circuit.

[2]We assume that wires within the same TAC have zero communication delay, while those connecting with other cells cost some non-negligible delays.

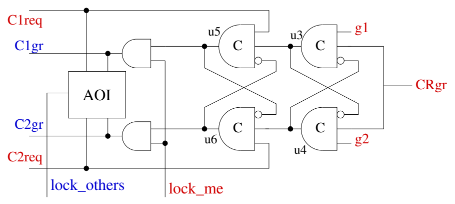

Fig. 9. Schematic incorporating the interlocking logic

The request of *C2*, which occurred at time instance t_2, reached the CRM at t_{12} due to long wt_{C1}. *C3*'s request, meanwhile, reached and acquired the CRM at t_{11} and t_{13} respectively. Without losing generality, the following condition, if true, will lead to unfair grants;

$$t_{11} < t_{12}$$

By substituting the values for t_{11} and t_{12} as depicted, and making few simplifications, such as, FL and BL of *T1* are approximately equal to that of *T2*, and rise and fall times of each gate are identical, the condition becomes;

$$t_8 < t_1 + 2(FL + BL) + wt_{C1}$$

In the worst case, *C2req* may occur at the same time with *C1req* and still lose the grant, i.e., $t_1 = t_2$;

$$t_8 < t_2 + 2(FL + BL) + wt_{C1}$$

where wt_{C1} may be substantial.

In contrast to above, the same condition using the proposed circuit is given by;

$$t_{11} < t_{17}$$

which upon simplification becomes;

$$t_8 < t_2 + 2(FL + BL)$$

Clearly, the unfairness window in the latter case is restricted to releasing and acquiring the arbiter in quick succession, and therefore guarantees relatively fairer arbitration.

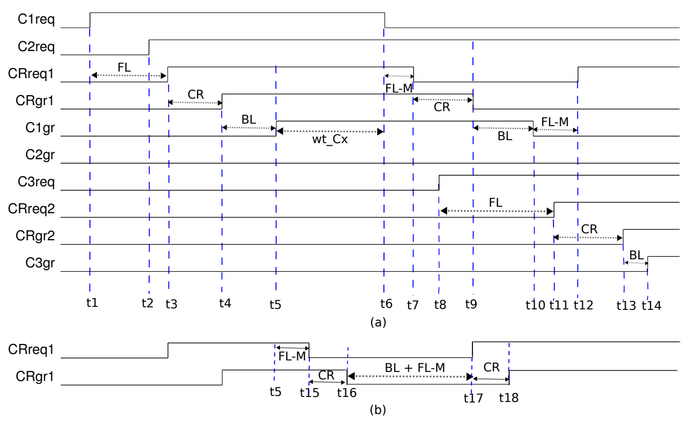

Fig. 10. Worst-case Unfairness Window: (a) TAC, (b) Proposed Circuit

IV. Implementation and Evaluation

We implemented four different designs for a four-clients setup, (a) standard TAC, (b) Ghiribaldi's 4-way arbiter [3], (c) SPA proposed in [6], and (d) the pipelined high speed TAC proposed in this work. All of the designs were synthesized for 90nm technology.

A. Worst and Best Case Latencies

Table I presents the latency of each arbiter with just one active client. Note that [6] presents a range of latencies; this is due to the different sized C-elements associated with each client: the lowest priority client has the largest C-element, and therefore, is the slowest as well. Something that is not apparent in the table is the fact that the cycle time for each design may be interpreted differently: For the first two, the given cycle times correspond to the best-case since this evaluation does not consider any delays due to the environment, otherwise the cycle time for each of them would linearly grow. For the remaining two designs, this time corresponds to the worst-case since both of them have some (virtual) pipelining employed, which would become visible when multiple clients were active and the delays of the environments were also considered. In simple words, increase in the cycle time (due to the environment) for one client, reduces *tphl* for itself, and *tplh* for the other client. This complementary behavior of the clients is not visible in our evaluation. For Felicijan's work, this shall have a marginal impact on $tphl$, roughly equivalent to saving a delay of a cascaded AND-gate and C-element pair. Refer to [6] for details.

TABLE I
Arbiters' latencies for only one active client

Design	tplh (ps)	tphl (ps)	Cycle time (ps)
STD TAC	401	266	667
Ghiribaldi	324	343	667
Felicijan	278 — 536	217 — 336	495 — 872
Proposed	400	548	948

Finally for our design, the given values hold true if only one client is active, and wt_{Cx} is as small as the delay of an inverter, which, as we have already argued, is extremely pessimistic.

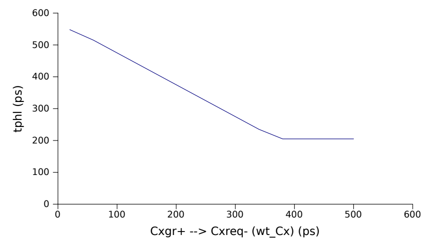

Fig. 11. Impact of increasing wt_{Cx} on latency of the proposed arbiter

In that case there is no slack time for our proposed pipelining to become effective. Fig. 11 depicts the improvement in $tphl$ with increasing wt_{Cx}. Starting from the delay equivalent to an inverter (corresponding to the worst case latency) up to a point around 360ps, $tphl$ falls almost linearly with wt_{Cx}. From this point onwards, the latency becomes constant (206ps), and is governed by the logic that we have added on top of the standard TAC. This adds to $tplh$ to represent the best case cycle time of 606ps for our design.

B. Handoff Latencies

Once again we assume that clients *C1* and *C2* share a local MUTEX, and *C3* and *C4* do the same with each other. This means that a handoff between *C1* and *C2* (in any direction) will be much slower than their handoffs with *C3* or *C4* for all the designs except for [6] in which H_{C1C2} shall be the fastest, and H_{C1C4} shall be the slowest. Similarly for all the designs except [6], handoffs between *C3* and *C4* will be slower than their handoffs with *C1* or *C2* on the other arbiter.

Table II summarizes these handoff latencies. For designs 1,2 and 4, the worst case latencies were computed by placing an inverter between *Cxgr* and *Cxreq* that would minimize wt_{Cx}. The best case latencies were computed by making wt_{Cx} longer than the arbiters' internal latencies. Note that the first two designs have identical best and worst case values. Because of its dependence upon wt_{Cx}, the results obtained for the proposed work are so diverging. Given an environment satisfying our assumptions, the proposed work can result in significantly faster arbitration, especially with all eager clients.

As far as design 3 is concerned, it is rather difficult to estimate the worst case latencies. In their design, the authors have used two variable environments: right hand side (rhs), and left hand side (lhs) of the arbiter, the former of which must be slower than the arbiter's internal latency in order for the design to work correctly. In our evaluation, that is how we computed the best case latency for this design. As a result, the worst case is determined by the upper bound on the rhs logic, which is obviously design specific.

To evaluate the threshold client's delay essential for the proposed methodology to allow high speed resource sharing, we have observed the behavior by gradually increasing wt_{Cx} from 20ps to 600ps, and plotting it against the average case latencies of other designs. Fig. 12 and fig. 13 present the handoff latencies between the clients on the same and different TACs respectively. It may be observed that beyond 200ps, the proposed methodology achieves the best throughput, which

TABLE II
HANDOFF LATENCIES

#	Design	best case $C1 \rightarrow C2$ (ps)	worst case $C1 \rightarrow C2$ (ps)	best case $C1 \rightarrow C3$ $C1 \rightarrow C4$ (ps)	worst case $C1 \rightarrow C3$ $C1 \rightarrow C4$ (ps)
1	STD TAC	613	613	355	355
2	Ghiribaldi	619	619	328	328
3	Felicijan	740	-	814 889	- -
4	Proposed	299	764	119	410

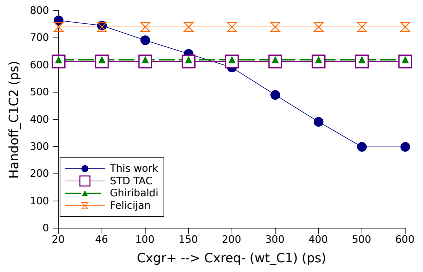

Fig. 12. Impact of wt_{Cx} on handoff latency on the same TAC

comes to saturation around 500ps mark. The saturation occurs since the elapsed time is sufficient for the virtual pipelining to have effectively completed its task in the background, and further delay of the client should not bring any improvement.

One advantage that the design 3 enjoys over the rest is its guarantee to not generate overlapping grants. In all other circuits, due to different rise and fall times of the standard gates, it may be possible that shortly before the grant to client 1 or client 2 has been removed, the grant to client 3 or client 4 is already set (this equally applies vice versa). Therefore all those designs require the clients to have some decoupling logic with the CR (that forces a null into the protocol no matter the grants are overlapping) to ensure safe handshaking. This additional logic will add a small performance overhead on designs 1, 2 and 4, which is not included in our analysis.

C. Throughput Estimation

To have a fair estimate of the best case throughput for each of the four designs considering equal priority traffic, we simulated two different orders of arbitration. In the first one, called *alternating* order, we made sure that the requests from the clients arrived simultaneously, leading to a *round* of arbitrations, in the order of grants G1, G3, G2, and G4 (therefore only observing the handoffs between the clients on different TACs). In the second case, called *sequential*, the requests from the clients arrived strictly in the order C1, C2, C3, and C4 (with sufficient delays in between, so that the effect of handoffs between the clients on the same TAC could also be observed), leading to the grants being given in the same manner as well (G1 $\rightarrow$ G4). Table III presents the throughput for each design corresponding to the two orders of arbitration, measured in Mega rounds (of arbitration) per second (Mrps). It is evident from the results that on average, the proposed work promises around 61.28%, 69.24%, and 186.85% higher throughput than the designs 1,2, and 3 respectively.

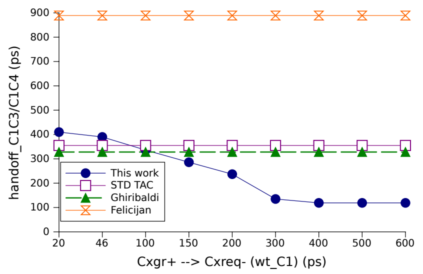

Fig. 13. Impact of wt_{Cx} on handoff latency between different TACs

TABLE III
COMPARISON OF THROUGHPUT

#	Design	Throughput (Mrps)	
		Alternating	Sequential
1	STD TAC	400	330
2	Ghiribaldi	384	312
3	Felicijan	206	206
4	Proposed	666	515

V. CONCLUSION

Arbitration is essential where resources need to be shared, and arbiter performance can have non-negligible impact on overall system performance. In this work we have proposed a novel tree arbiter cell that allows a pipelined processing of requests, i.e. arbitrating for the next request while the current one is still ongoing. The extra logic required for this feature initially increases the arbiter delay; however, in the relevant case of frequent requests from different clients our scheme yields a considerable speed-up. We have introduced an inter-TAC communication path for cascaded use of our TAC cell that not only enforces exclusive activation of a single grant at a time, but also improves the fairness of the arbitration process. Our simulation results clearly indicate that in most realistic cases our scheme provides superior performance; in an example NoC application we gained a speed-up of 61.28%, 69.24%, and 186.85% as compared to three different designs from literature. In environments where one client is more eager than the rest, designs 1 and 2, having the smallest cycle times, shall prove more useful than the proposed methodology.

REFERENCES

[1] D. J. Kinniment, *Synchronization and Arbitration in Digital Systems*. Wiley, 2007.

[2] A. Yakovlev, A. Petrov, and L. Lavagno, "A low latency asynchronous arbitration circuit," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 2, no. 3, pp. 372 –377, sept. 1994.

[3] A. Ghiribaldi, D. Bertozzi, and S. M. Nowick, "A transition-signaling bundled data noc switch architecture for cost-effective gals multicore systems," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, 2013, pp. 332–337.

[4] G. Dimitrakopoulos, N. Chrysos, and K. Galanopoulos, "Fast arbiters for on-chip network switches," in *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, 2008, pp. 664–670.

[5] C.-H. Huang, J.-S. Wang, and Y.-C. Huang, "Design of high-performance cmos priority encoders and incrementer/decrementers using multilevel lookahead and multilevel folding techniques," *Solid-State Circuits, IEEE Journal of*, vol. 37, no. 1, pp. 63–76, 2002.

[6] F. Feliciian and S. Furber, "An asynchronous on-chip network router with quality-of-service (qos) support," in *SOC Conference, 2004. Proceedings. IEEE International*, sept. 2004, pp. 274 – 277.