An Asynchronous Low Latency Arbiter for Quality of Service (QoS) Applications

Tomaz FELICIJAN, John BAINBRIDGE, *Member, IEEE* and Steve FURBER, *Senior Member, IEEE*

***Abstract*—In this paper we present the construction of a self-timed, multiple-input, priority arbiter with lower latency than existing solutions. The arbiter also overcomes the problem of allowing a contender to obtain over 50% of the resource allocation in a self-timed system by using downstream knowledge to trigger the arbitration. The arbiter is especially suited to the provision of quality of service in a self-timed interconnect.**

***Index Terms*—arbiters, asynchronous logic circuits, quality of service (QoS), on-chip networks**

I. Introduction

SYSTEM on Chip (SoC) denotes a methodology where designers combine pre-existing and/or new components and IP blocks on a single chip in order to achieve the required functionality of the whole system whilst reducing the overall costs and shortening time-to-market. Components of such systems are connected by an interconnect architecture which has replaced dedicated point-to-point connections.

A modern SoC application can implement various components with different traffic characteristics and constraints. For example, a video stream from a camera to an MPEG decoder requires high constant bandwidth but can tolerate relatively high latency, while an interrupt signal requires low bandwidth and low latency. It is therefore essential for an interconnect architecture to provide *Quality of Service* (QoS) capabilities in order to accommodate various components on the same network.

In essence, providing QoS requires reserving a certain proportion of network resource for a particular connection that demands preferential service. Those resources consist of buffer space and physical bandwidth. This paper investigates only the latter problem, reserving buffer space is not within the scope of the research presented here.

Reserving bandwidth in synchronous networks is usually done by *time division multiplexing* (TDM) [1] where the time axis is partitioned into time-slots each of which presents a unit of time when a single connection can transmit data over a physical channel. The bandwidth is reserved by dedicating a proportion of time-slots for a particular connection.

Manuscript received August 1, 2003. This work was supported by EPSRC under Grant GR/R47363/01.

The authors are with the University of Manchester, Department of Computer Science, Manchester, UK. (e-mail: felicijt@cs.man.ac.uk or jbainbridge@ieee.org).

In asynchronous networks the TDM technique is not applicable because it requires global synchronization between network elements. Another way to dedicate bandwidth is to employ a scheduling algorithm that will prioritise input requests according to the level of QoS required.

Figure 1 shows an example of three inputs competing for an output. The capacity of the output channel is 1 and inputs A and B require 1/2 and 1/3 of the available bandwidth respectively. Input C has no QoS demands (BE stands for best-effort). We assume that QoS inputs are not oversubscribed while input C is able to generate enough throughput to saturate the output.

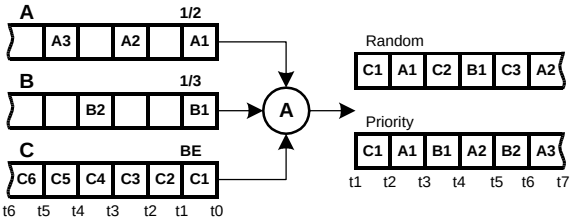

Fig. 1. Three input arbiter.

The departure events from the arbiter are depicted on the right side, and the arrival events are on the left. If we apply a random scheduling arbiter (upper right output sequence) the system does not provide the required bandwidth to inputs A and B and distributes the bandwidth randomly between the inputs. On the other hand a priority arbiter provides QoS inputs with the required bandwidth as shown in the figure. Inputs A and B acquire 1/2 and 1/3 of the bandwidth respectively, and input C acquires the remaining bandwidth (1/6). Note that the exact output sequence of the arbiter is impossible to predict due to the non-deterministic behaviour of the asynchronous circuit when multiple inputs arrive at approximately the same time.

II. Related Work

In the past the main concern when designing multi-way arbiters was to provide the property of fairness, meaning that when a request is issued it will be granted after a finite number of other requests have been granted. Token ring [2] and tree arbiters [3] both fall into this category. The sequence of the grants generated by such arbiters is non-deterministic [4], making them unsuitable for system-level design.

Priority arbiters based on the topology of the circuit, such as daisy chain and priority ring arbiters, provide a means of

controlling the sequence of grants. However, the worst-case latency of such systems grows linearly with the number of inputs, as does the implementation cost. Furthermore, a topologically fixed priority discipline makes these arbiters very inflexible and thus not sufficient to cover the wide range of modern applications.

Bystrov et al. presented a priority arbiter which operates in two stages [5]. In the first stage the arbiter locks the current state of the request vector in a special lock register comprising a two-way mutual exclusion element (MUTEX). At the second stage it computes a grant vector using combinatorial logic. Although the arbiter presents a very clever design, it suffers from two drawbacks. Firstly, the circuit is relatively slow with a period of approximately 40 inverter-delays and secondly, it cannot guarantee that a single input can acquire more than 50% of the available output bandwidth if multiple inputs are constantly arbitrating for the output.

The latter problem is common to most asynchronous arbiters and results from the fact that once a grant is released the arbiter starts arbitrating for the next output cycle immediately, leaving no time for the last granted input to recover and set the request signal high, thus giving the pending inputs a critical advantage to win the arbitration for the subsequent output cycle. A single input can therefore compete only for every other output cycle in the case when multiple inputs are constantly arbitrating for the output.

III. Proposed Solution

As noted above Bystrov's arbiter [5] implements a special register shown in figure 2 that locks the current state of the request vector until the grant is calculated. The register is controlled by a single input (lock) and generates a dual-rail [6] output (outputs w and l). When lock goes high the circuit sets one of the output signals to logic one, w if request signal r is active and l if it is inactive. The state of the output persists while lock remains high. When lock is set low and the request is removed the output goes low producing an empty dual-rail code-word.

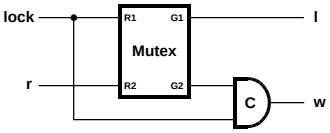

Fig. 2. A single bit lock register.

Furthermore, the register controls the start of arbitration by enabling signal lock. If request r arrives before lock is set high the register will not change the state of the output until lock is set low. Only when lock is enabled will the register produce a valid dual-rail output. This will activate combinatorial logic and the arbiter will generate the grant. Most arbiters will start the next arbitration as soon as the grant is released and at least one request signal is active. Bystrov's priority arbiter [5] follows the same behaviour because the positive edge of signal lock is generated by the 'ored' input request vector. Therefore, as soon as the grant is released the pending inputs will reactivate lock and restart the arbiter. This prevents the last granted input from competing for the subsequent output cycle as we mentioned before.

A. Principle of Operation

Our solution is based on the assumption that an arbiter is not the slowest part of a system and a critical section has a longer period than the arbiter. If this is correct we can decouple the arbiter from the critical section (CS) and delay the start of the arbitration to the last possible moment without sacrificing the performance of the system.

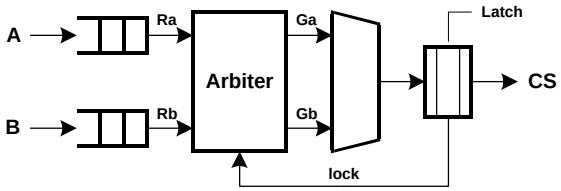

Fig. 3. Principle of the operation.

Figure 3 shows the principle of our approach. The system in the figure presents a mutually exclusive merge of two inputs into a single output. The arbiter is decoupled from the critical section by a latch and signal lock is generated by the output (rather than by the input as in [5]).

After the reset, lock is active and the system is awaiting data from the inputs. When at least one of the inputs arrives the arbiter generates the grant without any delay (apart from the delay inherent in the arbiter itself). This is normal behaviour because there is no way to know when the other input will arrive. After the output has been latched, lock is set low and the granted input is released. From this moment on, the arbiter and the critical section start to execute the current cycle independently with a speed that is limited only by the design of these two components. Note that if the other input has a request pending at the moment when the grant is released the arbiter will ignore that signal until the critical section has finished executing its current cycle and signal lock is set high. The behaviour of the system is denoted by the signal transition graph (STG) in figure 4.

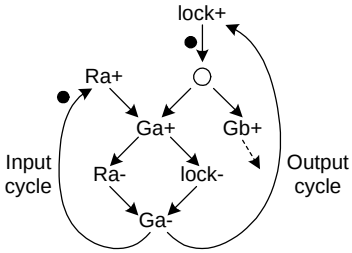

Fig. 4. Partial STG of the system.

The STG shows that if the input cycle, comprising the arbiter and the input FIFO, is fast enough to generate the new request before the positive edge of signal lock arrives, the input will be able to compete for every single output cycle providing there is enough throughput available at the input. If lock is activated before the new request is generated, the pending request (input B in our example) will win the

arbitration as indicated in figure 4 by the positive edge of signal Gb.

The lock signal is providing similar functionality in this four-phase design as the “done” signal in the classical two-phase request-grant-done (RGD) arbiter used by Sutherland in his Micropipelines work [7].

B. Implementation

Figure 5 shows a gate level circuit of a three input version of the arbiter with a linear priority module to calculate the grant vector. The priority module incorporates the C-element of a lock register (shown in figure 2) to reduce the latency of the circuit. The circuit does not include the output latch shown in figure 3. Signal lock is generated by a request vector (OR gate I4) and an enable signal (E) is basically an inverted acknowledge signal from the output latch. Asymmetrical C-elements I1...I3 prevent a MUTEX from being released before signal lock is deactivated during a return to zero stage of the arbiter. Inverted C-element I12 does not participate in the normal behaviour of the arbiter. Its function is to restart the arbitration when an empty request vector is locked.

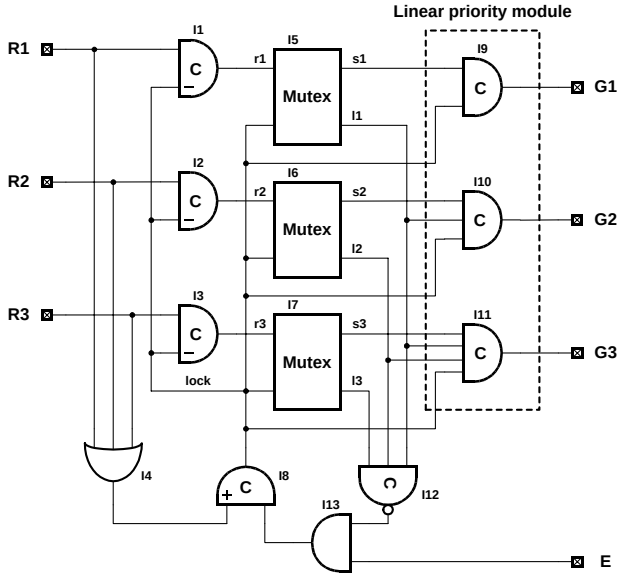

Fig. 5. Circuit of the arbiter.

The circuit is quasi delay-insensitive (QDI) [8] which means that it will operate correctly for arbitrary delays associated with the outputs of gates and mutually exclusive elements. However, the correct operation of the arbiter depends on isochronic behaviour of forks implemented in the circuit.

An STG in figure 6 describes the behaviour of the arbiter. Since all the inputs follow the same behaviour the STG shows the traces of only one input in order to simplify the graph and make it easier to follow.

1) Normal Operation

When at least one input is set high (R1...R3), OR gate I4 asserts signal lock through asymmetric C-element I8. Note that input E and inverted C-element I12 are both set to logic one after the reset. At the same time the request propagates through an asymmetric C-element (I1...I3) and sets signal r to logic one. Both signals compete for a MUTEX, but with reasonable wire layout, r should arrive first since it has to propagate through fewer, lighter loaded gates than the lock signal (I1 versus I4 and heavily loaded I8). On acquiring the MUTEX, r causes s to rise. Similar competitions occur for each MUTEX, to generate the inputs to the priority module that calculates a grant vector using one of the s signals and the l signals from the higher priority contenders.

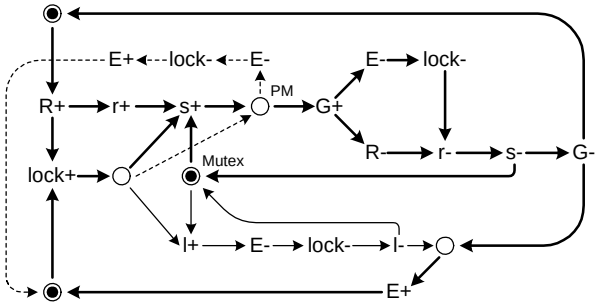

Fig. 6. STG of the arbiter.

After a grant vector is produced the arbiter has to wait for the environment to remove the granted request and set input E to zero. The asymmetric C-element at the input (I1..I3) only releases the MUTEX when r and lock are both low ensuring that the priority module does not generate an invalid grant vector. Furthermore, I8 ensures that lock is set low only after input E has been reset. When the MUTEX is released the priority module clears the grant and the arbiter is ready for the next cycle. Note that the next cycle will start only after input E has been set to a high level and not immediately after the grant is released. Bold arrows in the STG in figure 6 show this behaviour.

When a request (R2 or R3) is locked but not granted because the priority module made a decision to satisfy a higher priority input, the s signal remains high until the request is eventually granted. Note that signal r holds the state of the MUTEX while input E is low and lock is deactivated. This situation is marked with dashed arrows in the STG.

The third situation that can occur during the normal operation of the arbiter is when a request arrives after lock has been generated. When this happens the MUTEX sets output l to a high level which means that the particular request is not active and forwards this information to the priority module. The request has to wait until the next arbitration cycle when it will be sampled by the MUTEX as described by the following sequence: lock+, l+, R+, r+, E-, lock-, l-, E+, lock+, s+, (priority resolution), ...

2) Avoiding Hazardous Situations

The correct operation of the arbiter assumes that an electrical path from R to r is faster than a path from R to lock. We believe this is a fairly reasonable assumption for the reasons given above in section 3.2.1, and becomes even safer if the arbiter is extended to allow for additional contenders. However, to accommodate possible failures if this assumption does not hold (as a result of inadequate placement and/or

routing), gate I12 is included to detect the presence of an empty request vector locked into the MUTEX elements (signals l1...l3 are all set to one). In this situation, the priority module cannot produce a valid grant, and so gate I12 is used to cause a retry of the locking of the MUTEX elements which will resolve the situation and avoid a deadlock. This is undesirable since it increases the arbitration latency, but with an adequate circuit layout it should occur very infrequently or not at all.

The following sequence (not shown in the STG) illustrates this situation: R+, lock+, r+, l+, (deadlock detected), lock-, s+, (deadlock resolved), lock+, G+, ...

The other hazardous situation that could theoretically occur with this circuit due to poor layout and routing involves lock being deasserted as part of the clear-down phase after a successful arbitration won by R3. Lock falling causes l1 and l2 to fall, but if R2 is waiting, and l2 falls much quicker than l1, then C-element I10 could see s2 rise before l1 has fallen. To avoid this situation or others like it falsely triggering G2, the lock signal is used as an input to the gates in the priority module.

3) Arbiter limitation

The arbiter presented here has one limitation regarding the implementation of the priority module. A designer has to make sure that signals s1, s2 and s3 are only used to generate grant signals G1, G2 and G3, respectively. This is because signals s1, s2 and s3 can only be reset by input requests R1, R2 and R3, respectively.

As an example consider the function of G2=s1·s2. Once signal G2 is asserted it will never be reset because signal w1 will not go low until request R1 (which has set s1 high) is removed. Unfortunately this cannot happen because the output is stuck at G2 high and the system deadlocks.

IV. Conclusions

Providing QoS in asynchronous systems is not a trivial task and requires careful design both at the circuit and the system level. There are several timing constraints that have to be met in order for the system to operate inside the boundaries of the specifications. The arbiter presented in this paper was designed to loosen those constraints and to provide designers with a fairly deterministic asynchronous building block.

The arbiter presents a low latency solution with an approximately 20 inversions period (the period depends upon a priority module implemented in the design) and the ability to guarantee a throughput of more than 50% of the system's bandwidth for a single input, the feature that has not been available with the arbiters that are known to us. The circuit is quasi-delay-insensitive (QDI) and will operate correctly for arbitrary delays of gates and wires but it will provide a certain level of QoS only when several timing constraints are fulfilled as described in section 3.

References

[1] ATM Forum. http://www.atmforum.com.

[2] A. J. Martin, "The Design of a Self-timed Circuit for Distributed Mutual Exclusion," *In 1985 Chapel Hill Conference on VLSI*, pp. 245-260, (1985).

[3] M. B. Josephs and J. T. Yantchev, "CMOS Design of the Tree Arbiter Element," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(4), pp. 472-476, December 1996.

[4] C. Seitz, "System Timing", Chapter 7 of *Introduction to VLSI Systems* by C. Mead, L. Conway, Addison Wesley Second Edition, 1980.

[5] A. Bystrov, D. Kinniment and A. Yakovlev, "Priority Arbiters," *Int. Symposium on Advanced Research in Asynchronous Circuits (ASYNC)*, Eilat, Israel, pp. 128-137, April 2000.

[6] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design: A System Perspective*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 2001.

[7] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, 32 (6), pp. 720-738, June 1989.

[8] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," In W. J. Dally, editor, *Sixth MIT Conference on Advanced Research in VLSI*, pp. 263-278, MIT Press, (1990).