Department of Computing Science,
University of Newcastle upon Tyne

Priority Arbiters

A. Bystrov, D. J. Kinniment, A. Yakovlev

Technical Report Series

No. CS-TR-687
October, 1999

Contact:

a.bystrov@ncl.ac.uk

David.Kinniment@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

Published by the University of Newcastle upon Tyne,
Department of Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, UK

Priority Arbiters

A. Bystrov, D. J. Kinniment, A. Yakovlev

October 22, 1999

Abstract

This report presents asynchronous design solutions to the problem of Priority Arbitration which is defined in the following form. A system consists of multiple, physically concurrent, processes with a shared resource. The discipline of resource allocation is a function of parameters of the active requests, which are assigned to the requests either statically or dynamically. This function can be defined in an (arbitrary) combinatorial way (contrary to conventional, 'topological', mappings, such as that used in a daisy-chain arbiter).

The proposed designs are truly delay-independent. Furthermore, the priority logic, in the dynamic case, has the following architectural feature: it is a tree structure in which the control flow is maximally decoupled from the data-path by means of an early propagation of the 'valid'-'invalid' signals, concurrently with processing the priority data. This leads to significant reduction in the overall arbitration delay when the number of active requests is low.

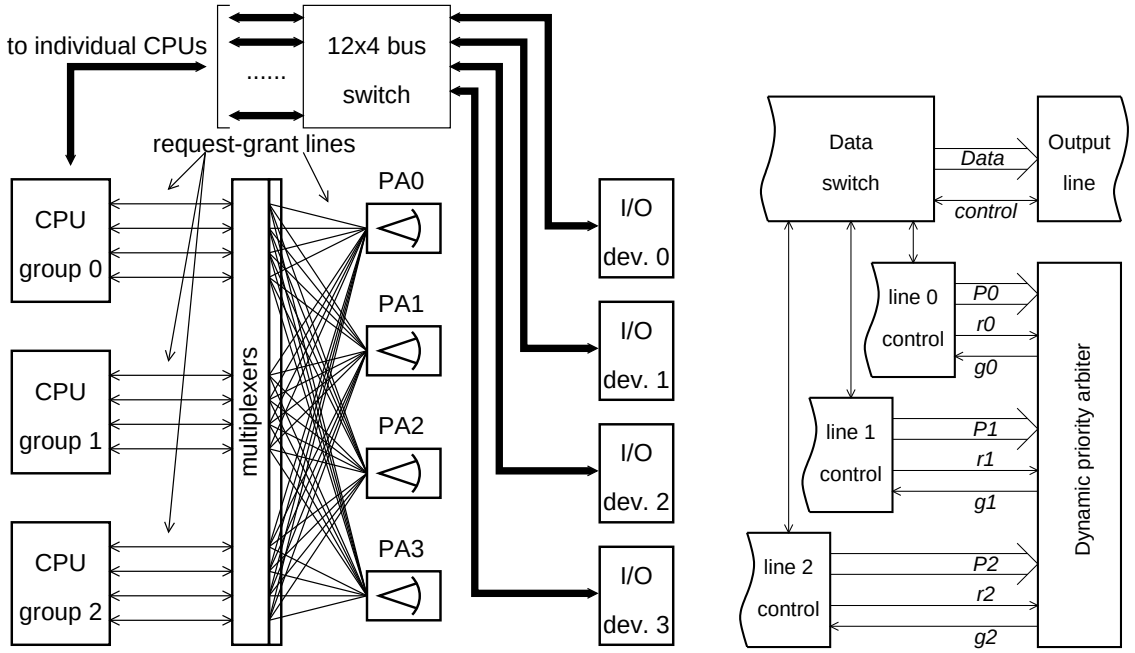
Keywords: arbitration, asynchronous systems, priority resolution, low-latency design.

1 Introduction

The advance of VLSI technology increasingly results in a large part of a system being placed on a single chip. The blocks of such a system communicate via very fast local signals and require an adequate means of controlling their access to shared resources [1]. In traditional multi-way arbiters [2][3], such as token ring, mesh and tree arbiters [4], the main concern is to provide a property of 'fairness' [6], which guarantees that a request is granted after a finite number of other requests are granted. Little or nothing is done to maintain a discipline of access which could be useful for the system-level design. Disciplines of grant calculation, such as fixed linear priorities implemented in 'daisy chain' arbiters and a priority ring implemented in a token ring arbiter, are not sufficient to cover the range of modern applications. In the recent paper [7] we have proposed an 'ordered arbiter' which keeps the order of grants as close to the order of request arrival as possible. Its algorithm is implemented in hardware as the speed of software implementations is often not sufficient for system-on-chip designs. In this report we propose two arbiters which use priorities to generate grants.

Traditional applications of arbiters with simple priority disciplines such as linear priorities and a priority ring are well known. In this report we introduce the enhanced

formulation of a priority discipline, which is understood as grant calculation based on the current state of the request vector. Examples in Fig. 1 show some applications of arbiters with enhanced priority disciplines.



(a) Shared port access (b) Network priority switch

Fig. 1: Applications of priority arbiters

The example in Fig. 1(a) shows a system comprising three groups of processors (every group can be represented by a single multithreaded CPU) sharing four I/O devices. The nature of the task executed by every group is such that an individual CPU may be suspended waiting for calculation results from the other members of its group. Further, when the number of pending requests from the group increases, the probability that an individual CPU becomes suspended drastically increases. To minimise average waiting time it would be beneficial to grant I/O access to the group with the largest number of pending requests. This can be done by taking a 'snapshot' of the request bus at the moment of arbitration and calculating the grant as a function of the request bus state. The priority arbiters PA0 ... PA3 perform this task. Their possible implementation and properties are discussed in Section 3.

A fast network priority switch in Fig. 1(b) is another example of the advanced priority discipline. It has three input channels requesting access to a single output port. Every request is complemented with a priority value transmitted through a dedicated priority bus. Priority values (attributes of requests) are generated dynamically. A dynamic priority arbiter takes a 'snapshot' of the request bus (which is similar to the previous example) and calculates grant as a function of the request bus state and the state of those priority buses which are accompanied with the active request. Realisations of such arbiters are described in Section 4.

The main contribution of this report is threefold.

Firstly, the priority discipline of an arbiter is formulated as a combinational function defined on the current state of request inputs, which is less restrictive than conventional, 'topological', mappings, such as that used in a daisy-chain arbiter.

Secondly, the design of the priority module is based one a tree structure, whose special feature is the use of a three-signal control ('valid'-'invalid'-'done'). This allows the system to maximally decouple control flow from the data-path by means of propagating the 'valid'-'invalid' signals concurrently with processing the priority data. The datapath computation can be 'disregarded' (depending on the 'valid'-'invalid' inputs) and grant issued with a very low latency. This acceleration significantly reduces the overall arbitration delay when the number of active requests is low.

Finally, the circuits presented in this report are designed to operate correctly for arbitrary delays associated with the outputs of gates and mutual exclusion elements. This property is called delay-independence; it is 'almost' identical to speed-independence. We, however, avoid using the latter term (originating from Muller's work [6]) because it is traditionally applied, in a more formal context, to circuits that only consist of logic gates (without mutexes).

2 Priority discipline

All arbiters exhibit non-deterministic behaviour. When several requests arrive simultaneously (or almost simultaneously), only one becomes granted and this choice is random. However, arbiters differ in handling pending requests. In a *simple* arbiter, such as a 2-way arbiter based on cross-coupled logic gates, pending requests are arbitrated after the currently granted request is removed. This causes secondary metastability [7]. *Complex* multi-way arbiters (comprising simple arbiters) usually perform arbitration of pending requests during the critical section in the process which is currently granted. The arbitration results are stored in the internal memory and are used to compute a grant after the currently processed request is reset.

The method of grant computation can be used for classifying complex arbiters. If such a computation takes into account the history of request processing, then an arbiter is said to exhibit *sequential* behaviour. Token ring, mesh and tree arbiters fall in this class. If the computation is entirely based on the current state of the request vector, then an arbiter is *combinational*. The combinational function of grant computation can be understood as a *priority discipline*. This permits us to refer to a combinational arbiter as a priority arbiter (PA).

The following types of priority discipline can be identified:

- topologically fixed;
- static with an arbitrary priority function;
- dynamic.

A *topologically fixed priority discipline* is observed in a daisy-chain arbiter realising linear priorities. The 2-phase STG in Fig. 2 shows that the priorities are defined by the order of request polling. This order cannot be changed without changing the topological structure of the system.

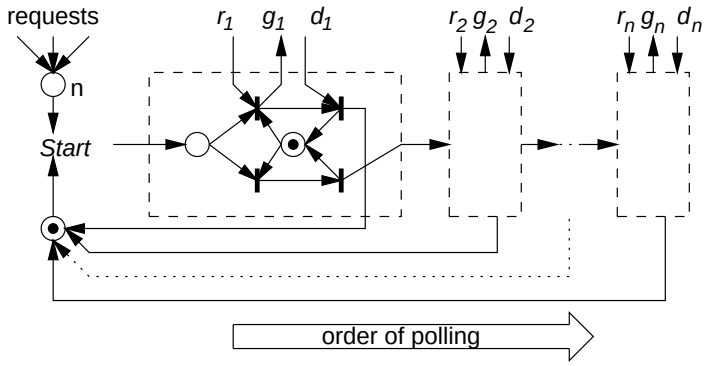


Fig. 2: Daisy-chain arbiter STG

A *static priority discipline* with an arbitrary priority function is implemented in the static priority arbiter (SPA) shown in Fig. 3 and described in Section 3. In this arbiter the fixation of the request vector is completely separated from the combinational circuit realising the priority function. The priority function can be changed by modifying the priority combinational circuit without altering the arbiter structure. Further, this can be done at the initialisation time by setting up additional inputs of the priority circuit, thus making the arbiter programmable. An SPA can implement any priority function as opposed to an arbiter with a topologically fixed priority discipline, which is defined by the order of request polling. For example, if request lines form several groups, it is possible to define such a priority function that always gives a grant to a member of the group with the largest number of requests. Another example is context-dependent arbitration which gives a higher (or lower) priority to a request that belongs to the group of active requests forming a particular pattern.

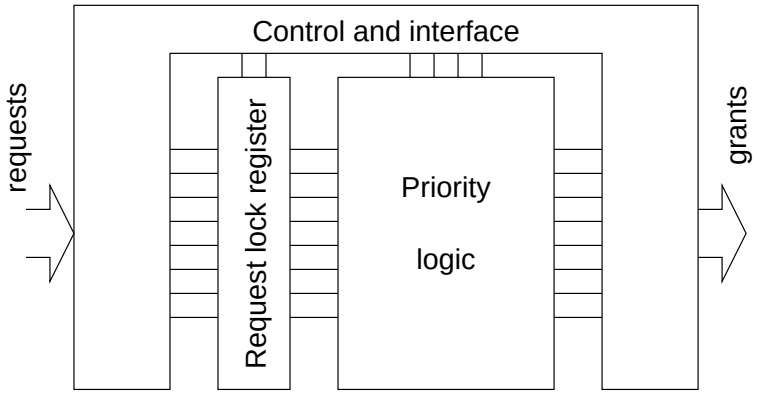


Fig. 3: Static priority arbiter

A *dynamic priority discipline* is different from the static one as it uses dedicated priority buses to receive priority information from clients. Fig. 4 shows a dynamic priority arbiter

(DPA) described in Section 4. The most obvious application of the dynamic priority discipline is to detect a request with the highest priority value on the priority bus.

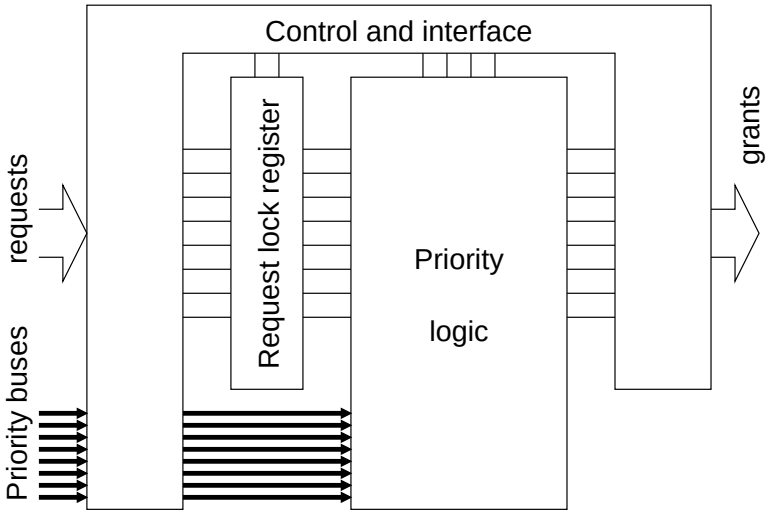


Fig. 4: Dynamic priority arbiter

3 Static priority arbiter

The structure of the proposed SPA originates from the idea described in [8]. It operates in two stages. At the first stage it locks the request vector in a register comprising 2-way arbiters (MUTEX elements). At the second stage it computes the grants using a combinational priority module. The separation of arbitration and grant computation functions allows the circuit to achieve truly combinational behaviour. Interface and control logic provides the correct functionality of the device under arbitrary gate delays.

A single bit of the lock register is shown in Fig. 5. It is similar to the request polling block of the token ring arbiter [4]. The output of this circuit forms a dual-rail code, in which code words are separated by the all-zero spacer word. This facilitates the design of the priority module as a dual-rail circuit. Note, that for a speed-independent realisation the AND gate in Fig. 5 must be replaced by a C-element, or other appropriate means must be implemented to indicate the reset phase of the signal s.

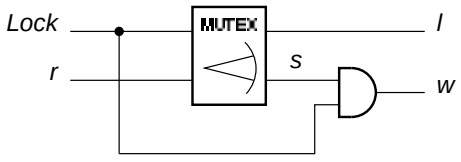


Fig. 5: A bit of the lock register

A delay-independent realisation of SPA is shown in Fig. 6. The input requests R_{1_3} propagate through the set-dominant latches I1 ... I3, having function $q^{\uparrow} = set + q \cdot \overline{reset}$, to the MUTEX elements I7 ... I9. After at least one of these reaches a MUTEX, a high signal level is formed at the output of the reset-dominant latch I16, having function

$q^{\uparrow} = \overline{reset} \cdot (set + q)$. This signal (*Lock*) is applied to the upper inputs of **MUTEX** elements, thus locking the request vector. After the input vector is locked, a dual-rail signal is formed at the inputs of the priority module. The priority module is implemented as dual-rail logic to guarantee, that the output signal is generated after the process in the **MUTEX** elements is settled. The priority module determines which request should be granted and raises the signal at the appropriate output. This signal becomes locked in the output buffer comprising C-elements (**I20** ... **I22**).

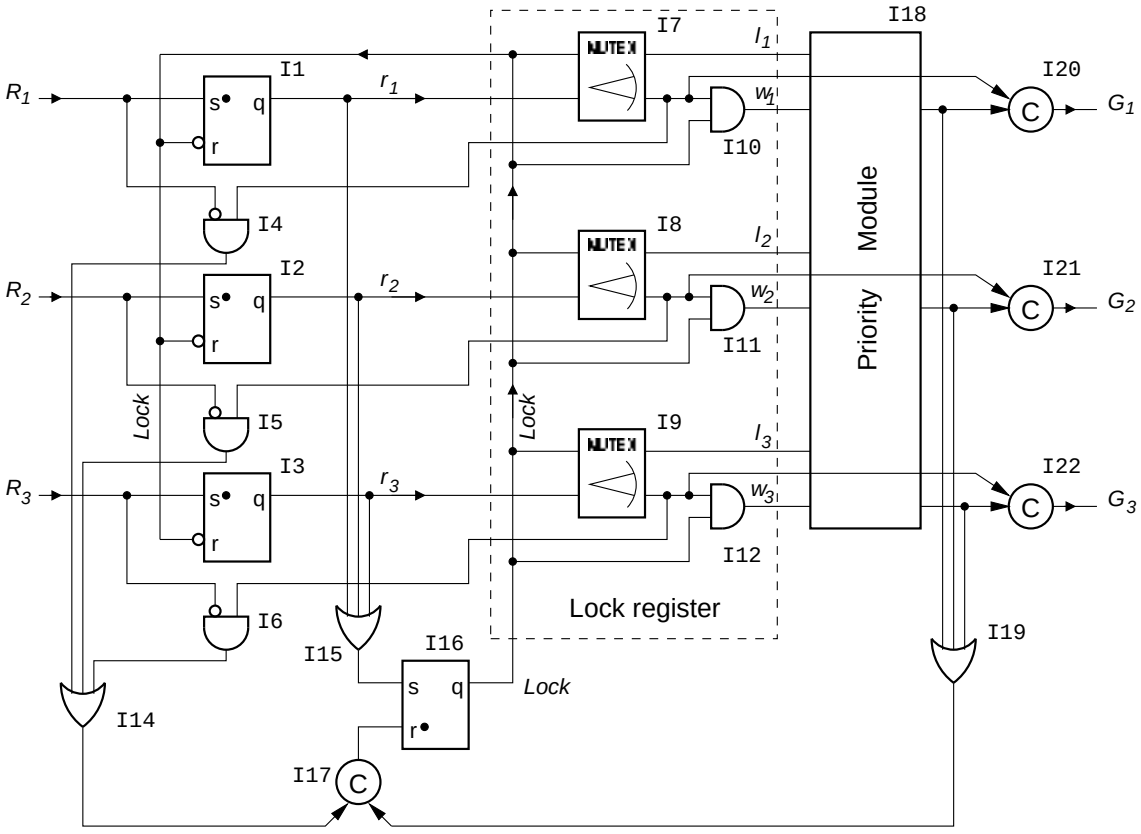


Fig. 6: 3-way SPA

After using the resource, the client resets its request. This is detected by **I4** ... **I6** and causes switching of **I17** and **I16**, resulting in the resetting of the *Lock* signal. This resets the input latch, the **MUTEX** in the channel where the request has been dropped and all input signals of the **PM**. This resets the **PM** outputs and the grant signal. Then high levels are formed at the outputs of **I14** and **I19**, which switches the C-element **I17**, unblocks **I16** and enables processing of the next request.

Correct operation of the lock register is crucial for the priority arbiter. An STG in Fig. 7 shows how the polling cell is made insensitive to the gate delays. This STG describes the behaviour of a single channel, which explains why the indices after R, r, s, l, w and G signal identifiers corresponding to the request channels are not shown.

During its operation, the arbiter produces the following traces corresponding to the STG in Fig. 7. If, at the moment of the *Lock* setting, the request R is not present, then the trace *(Lock+, l+, Lock-/1, l-)* is produced, which means that the polling cell has

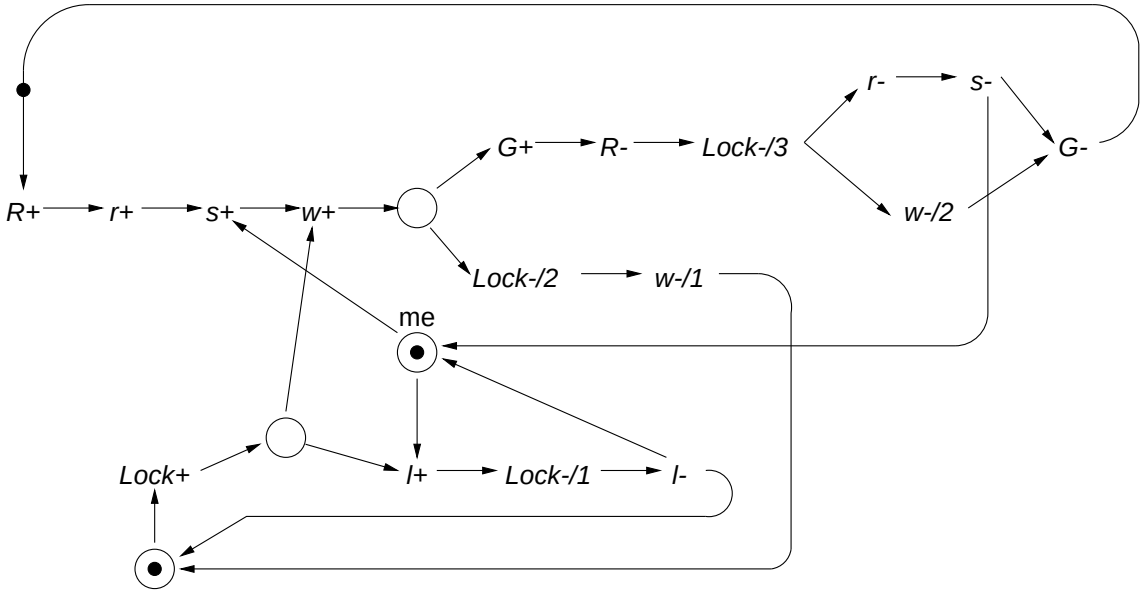


Fig. 7: STG of the request lock register

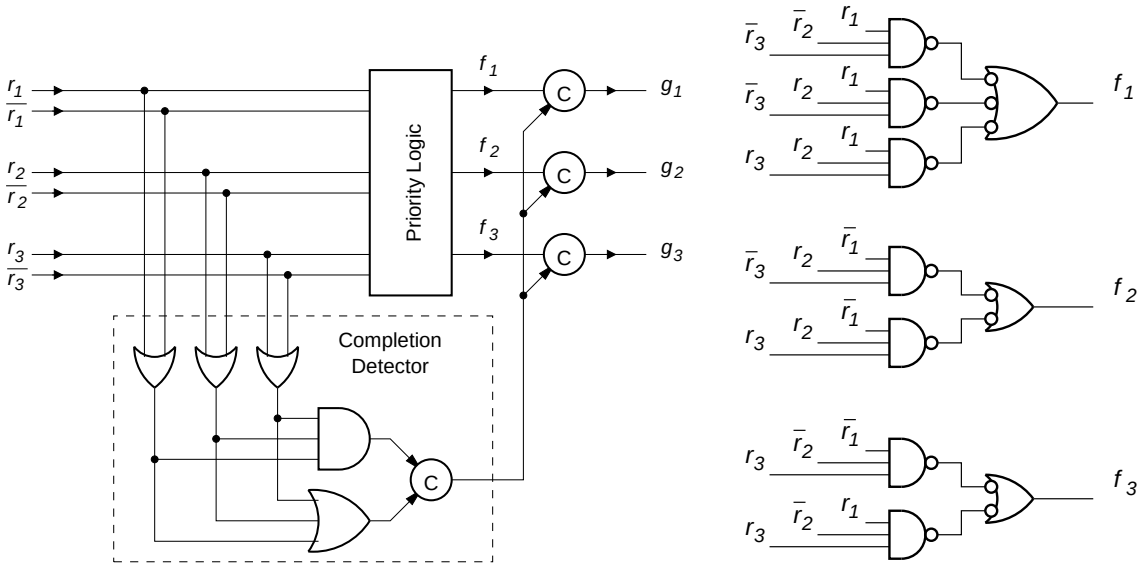
registered a 'lose' situation and the output grant has not been issued. If, at the moment of the *Lock* arrival, the request R is present, then two situations may take place. The first is described by the trace (R+, r+, s+, *Lock*+, loop:{w+, priority_resolution, *Lock-/2*, w-/*1*, *Lock*+}), which covers the scenario when the request is locked (s+), but the grant G is not issued. This happens if the **PM** makes a decision to satisfy a higher priority request in another channel that is not shown in Fig. 7. The trace part labelled as 'loop' is executed repeatedly until the request R is satisfied. The second situation, in which the **PM** satisfies the request R, is described by the trace (R+, r+, s+, *Lock*+, w+, priority_resolution, G+, R-, *Lock-/3*, r-, s-, G-). These traces clearly show that a dual-rail code is formed by the polling cell at its outputs w and l every time after a *Lock*+ event. The code words are separated by all-zero words (w=0 and l=0), which facilitates the use of the dual-rail **PM**.

In the arbiter circuit shown in Fig. 6, the only gate output which is not properly acknowledged is that of the OR gate I15. It can either be considered as part of a complex gate inside the I16 latch, or its delay must be bounded by a reasonable value.

A possible dual-rail realisation of the **PM** is shown in Fig. 8. The **PM** produces 1-hot output after the dual-rail input is settled. The output becomes reset to an all-zero input word (separator). The hazard-free priority logic (**PL**) can be implemented as sum-of-products circuit without inverters as shown in Fig. 8(b). The truth table of this **PL** shown in Table 1 implements a linear priority system.

Symbol w in Table 1 means 'win' and signifies the request locked in the **MUTEX** register. Symbol l means 'lose' and indicates the request which is considered to be absent at the moment of arbitration.

A more complex example of the priority function is shown in Table 2. It handles requests that form two groups $group_1 = \{r_1, r_2\}$ and $group_2 = \{r_3, r_4\}$. The higher priority is always given to a group with the largest number of active requests. If the



(a) dual-rail priority module (b) hazard-free logic

Fig. 8: Priority module

input			grant
r_1	r_2	r_3	g_i
1	1	1	not possible
ur	1	1	g_1
1	ur	1	g_2
1	1	ur	g_3
ur	ur	1	g_1
1	ur	ur	g_2
ur	1	ur	g_3
ur	ur	ur	g_1

Table 1: Linear priority function

number of requests is equal, then $group_1$ wins. Within a group a request with the lower number has greater priority.

This priority function can be implemented as a sum-of-products hazard-free circuit similar to the one shown in Fig. 8(b).

4 Dynamic priority arbiter

A dynamic priority arbiter (DPA) considered in this section performs comparison of the priorities supplied with each request and grants the one with the highest value. The structure of DPA shown in Fig. 4 is very similar to the structure of SPA. The difference is that DPA uses additional data (priority values) produced by clients and passed into the DPA through dedicated priority buses. The value on a priority bus is valid only when the corresponding request is present. This creates a problem with speed-independent realisation

input				grant
$group_1$		$group_2$		
r_1	r_2	r_3	r_4	g_i
l	l	l	l	not possible
ur	l	l	l	g_1
l	ur	l	l	g_2
l	l	ur	l	g_3
l	l	l	ur	g_4
ur	ur	l	l	g_1
l	ur	ur	l	g_2
l	l	ur	ur	g_3
ur	l	l	ur	g_1
ur	l	ur	l	g_1
l	ur	l	ur	g_2
ur	ur	ur	l	g_1
l	ur	ur	ur	g_3
ur	l	ur	ur	g_3
ur	ur	l	ur	g_1
ur	ur	ur	ur	g_1

Table 2: Priority function supporting groups

of the priority module as a delayed transition on the priority bus may be interpreted as an incorrect value. Another problem is to design a comparator for k n-bit priority buses which takes into account only the values accompanied by a request. Priority bus state disregarding is a non-trivial task in a speed-independent design as special means are required to prevent hazards. In both the speed-independent and the bundled-data approaches a special signaling is needed to distinguish the data which have not arrived yet due to the datapath delay from the data will not arrive because the corresponding request is not set.

The first problem is solved by using a dual-rail or a 1-hot code on the priority buses of the delay-independent arbiter realisation. This is not needed for the bundled-data realisation. The second problem is solved by introducing two signals v ('valid') and i ('invalid') which accompany every priority bus. These signals are generated by the lock register (similarly to ur and l signals in the SPA) and propagated through the tree comparator distinguishing the priority value which has not reached the node yet from the absent request.

A delay-independent DPA realisation with 8 requests and 4 request priority values is shown in Fig. 9. Priority buses carry a dual-rail code, though a design with 1-hot code has also been considered. The lines of the dual-rail priority buses $P0\langle 0:3\rangle$ ... $P7\langle 0:3\rangle$ are encoded as following: $P_i<0>$ is the lower bit, $P_i<1>$ is the higher bit, $P_i<2>$ and $P_i<3>$ are complimentary bits defined as $P_i<2> = \bar{P}_i<0>$ and $P_i<3> = \bar{P}_i<1>$. Gate arrays in this figure are labelled as x4 or x8 depending on the number of array elements. If a bus is connected to such an array, then every bus line is assumed to be connected to the gate with the index number corresponding to the index number of the line. The lower part of the DPA schematic is similar to SPA. The upper part, comprising

priority buses, AND gate arrays and a reset completion detector is new.

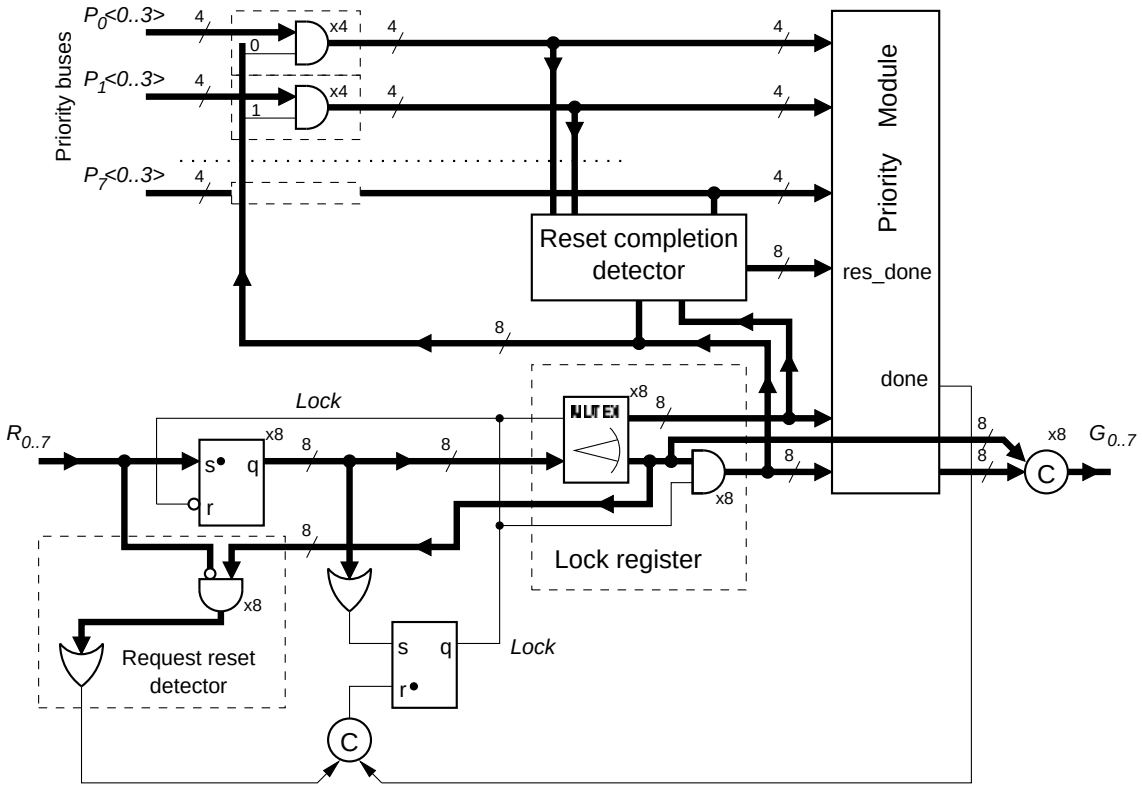


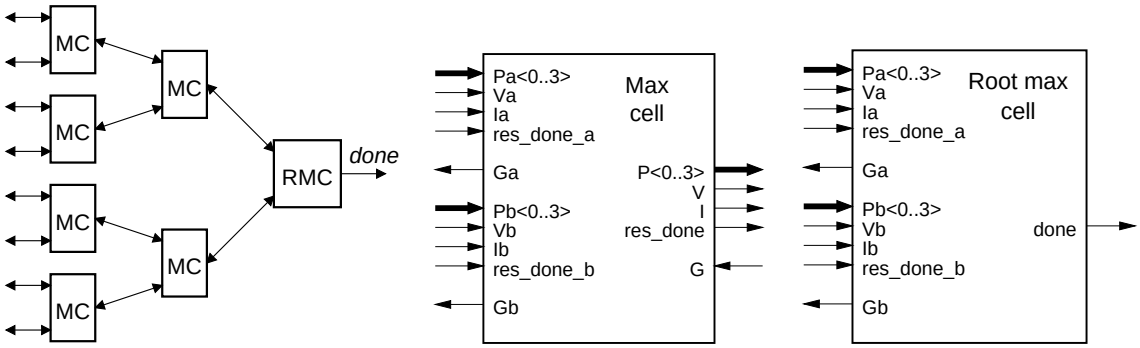
Fig. 9: Dynamic priority arbiter

Several aspects enhancing functionality and improving performance were considered in the design of this arbiter. Arbitration and grant computation are separated, which permits realisation of an arbitrary priority discipline. The control flow and the priority data flow are maximally decoupled in the priority module. This permits control signals to run forward without waiting for the slow data if the data are not needed to compute the result. For example, if a single request arrives, then the output of the priority module and the grant are generated almost immediately. Only the *done* signal waits for the completion of data manipulation, which happens concurrently with a critical section in the client process. Handshake phases in the control and data flow are also decoupled. C-elements are replaced by AND gates where possible and a reset phase of their inputs is indicated by additional 'reset-done' detectors.

The operation of the control part of the DPA in Fig. 9 is similar to SPA. The AND gates on the priority buses are only needed in the speed-independent realisation to provide an all-zero separator word for the dual-rail priority module. The separator word is indicated by the reset completion detector, which also checks valid-invalid signals for every channel. This detector comprises eight 6-OR gates (one per channel). The inputs of every gate are connected to the lines of a priority bus and to the corresponding valid-invalid signals. Its output goes high as soon as the set phase in the channel begins (the completion of the set phase is indicated inside the priority module). The output goes low only after all inputs are reset establishing the separator word. The outputs of the reset completion detector are

connected to the inputs of the reset completion system of the priority module.

The structure of the priority module is shown in Fig. 10. It uses a tree structure shown in Fig. 10(a) to find the maximal priority value. Every node of the tree is the maximum calculation cell (MCC) shown in Fig. 10(b). The root MCC (RMCC) is different from others and its symbol is shown in Fig. 10(c).



(a) Tree structure of maximum calculation priority module

(b) Maximum calculation cell (MCC)

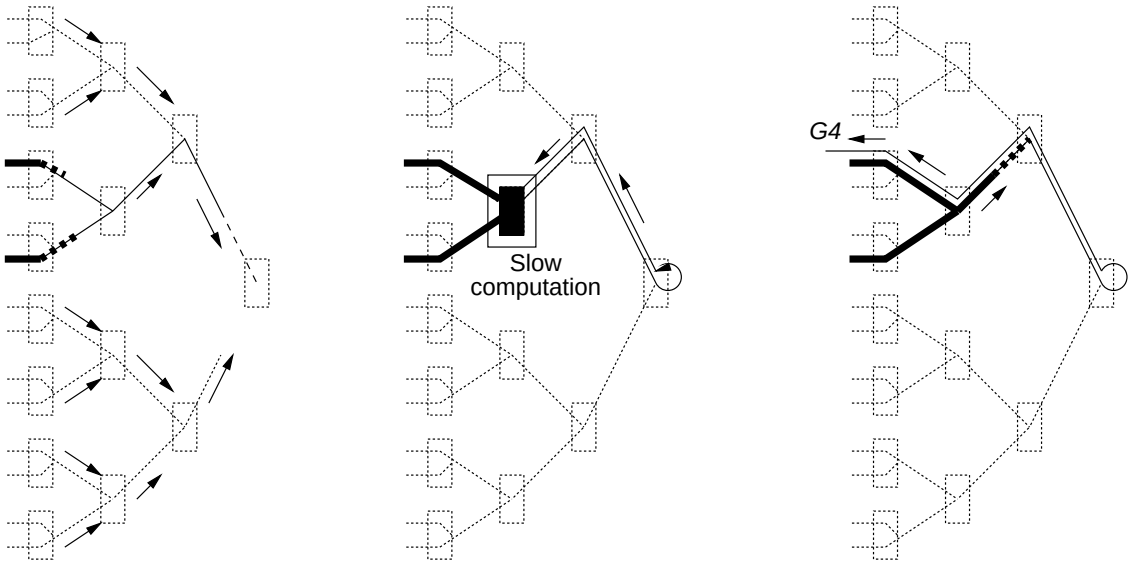
(c) Root maximum calculation cell (RMCC)

Fig. 10: Structure of the priority module

Every cell has three modes of operation. If it gets 'invalid' signals in both channels, then the output 'invalid' signal is issued. All input priority data are disregarded and the output priority bus is set to the all-zero spacer word. If only one channel is 'valid', then its priority data are transferred to the output bus and the internal comparator is not invoked. If both channels are 'valid', then a comparator is used to compare the priority values and the greatest of the two is propagated to the output. In the last two cases the output 'valid' signal is generated.

The control 'valid' signal is permitted to propagate through the tree structure without waiting for the corresponding data. As a result these signals reach the root node very quickly and may even cause the RMCC to generate a primary grant (Ga or Gb). This happens if the RMCC receives a 'valid' signal in one channel and 'invalid' in the other. Further, such an accelerated grant may start propagating backwards if the nodes it propagates through do not need data to make a decision. Synchronisation with datapath takes place only if the priority value is needed at the given MCC to decide to which MCC port the grant should be forwarded. This acceleration technique is illustrated in Fig. 11. The nodes of the tree-type maximum calculation priority module are depicted as dotted rectangles. Thick lines show propagation of the priority data. Thin solid lines depict active 'valid' signals and dotted lines depict active 'invalid' signals. Arrows show the direction of signal propagation at the given stage.

After 'valid' and 'invalid' control signals are produced by a request lock register, they begin to propagate through the maximum calculation priority module as shown in Fig. 11(a). Their propagation is faster than the propagation of the priority data. A primary grant in Fig. 11(b) is generated by the root node without waiting for data, because having one



(a) Fast control propagation

(b) Grant propagation during data comparison

(c) Early grant output

Fig. 11: Accelerated grant generation

channel 'valid' and another 'invalid' is sufficient to decide which to grant. Then the primary grant begins reverse propagation to the nearest node which has both channels 'valid'. Only a node performing data comparison can delay grant propagation. After finishing the computation, the node permits grant propagation and at the same time transfers the maximum priority value to the output priority bus as shown in Fig. 11(c). This figure illustrates that a grant can be released before priority data reaches the root node.

This acceleration technique is similar to that used in a low latency tree arbiter [9]. The difference is that in the described system slow events are associated with dataflow. To minimise latency the dataflow and the control part ('valid'-'invalid' signals) are maximally decoupled. Synchronisation is performed only when the data are needed to make a decision. This results in better performance based on a higher degree of concurrency.

The accelerated propagation of control and grant signals does not allow the indication of the set phase on the priority bus. Without this, one cannot guarantee a proper indication of the next phase, which is a reset. This problem is solved by using dual-rail (or 1-hot) circuits as comparators incorporated in MCC. As a result, a value on the priority bus at the root node becomes a code word (dual-rail or 1-hot) after the set phase on all 'valid' buses is completed. A completion detector in the root node generates a 'data_done' signal, which is used to generate the output *done*.

Additional signals 'reset_done' shown in Fig. 10(b,c) indicate the reset phase on 'valid'-'invalid' lines and priority buses and allow to replace C-elements by the faster AND-gates in the node circuits. This also accelerates the resetting of the priority module because the control signals causing the reset are propagated without any synchronisation at this stage

(the cause is separated from the indication).

A possible realisation of the MCC is shown in Fig. 12. Gates I18 ... I21 calculate the mode of operation. If both input channels are 'valid' ($Va = 1$ and $Vb = 1$), then I18 generates a low level at its output, which causes the output $V = 1$ ('valid') signal and connects the comparator (I6) inputs to the input priority buses. When the comparison of the priority values is finished, a dual-rail output signal performs switching of the request bus multiplexer I8 and the grant demultiplexer I22-I23. If both input channels are 'invalid' ($Ia = 1$ and $Ib = 1$), then I19 and I25 generate the output 'invalid' signal $I = 1$. If one channel is 'valid' and another 'invalid' ($Va \wedge Ib = 1$ or $Ia \wedge Vb = 1$), then I20 or I21 produces a low level, switching the priority buses and grant lines without invoking the comparator.

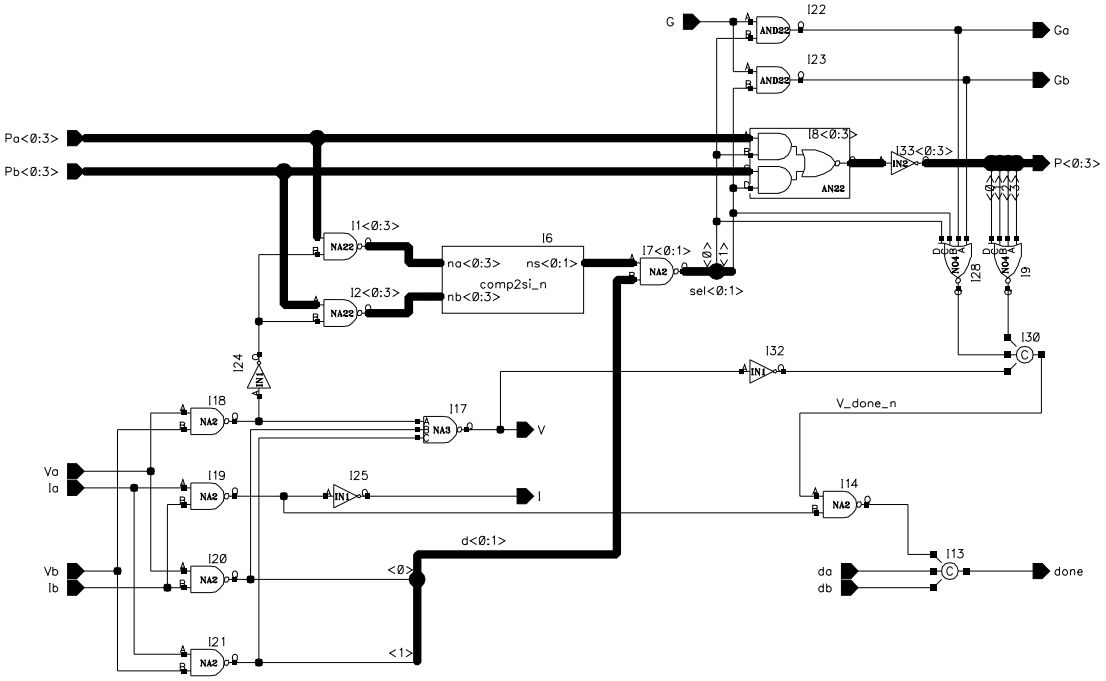


Fig. 12: Maximum calculation cell

All control circuits and data switches are implemented as combinational gates, which requires special means to indicate their reset phase. The reset detector is implemented as I9, I14, I28 and I30 gates, though it could be a single 10-input OR gate if such were available in the standard library. The output of the reset detector is synchronised by I13 C-element with the input signals *da* and *db* indicating reset in the previous MCCs. Thus the output *done* signal is produced. The advantage of such realisation is the extremely fast propagation of the control 'valid'-'invalid' signals to the V and I outputs, which requires just two simple gate delays. The comparator is invoked only when it is needed.

The realisation of the RMCC is shown in Fig. 13. It differs from the MCC realisation in having completion detectors on priority buses and having no V and I outputs. The RMCC terminates any path of priority data propagation. This requires a proper indication of all handshake phases on the priority bus (as opposed to the reset phase only indication in

MCCa). The completion detectors comprise gates I3, I4, I36, I37, I38 and I39. The output signal *done* indicates the completion of all processes in the datapath and the control logic, thus providing hazard-free operation of the system under arbitrary gate delays.

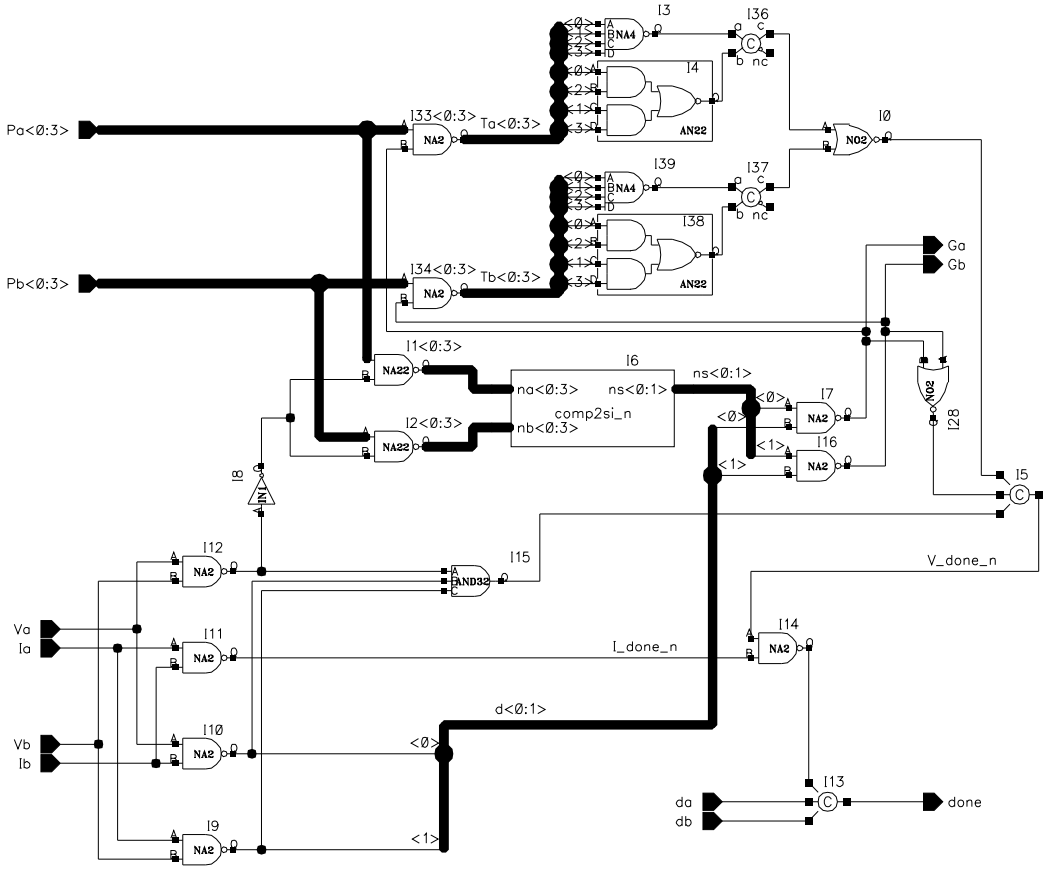


Fig. 13: Root maximum calculation cell

The structure of the completion detectors in Fig. 13 corresponds to the dual-rail code. It is simpler for the 1-hot code, which requires just a single OR gate to indicate all handshake phases.

The comparator is realised as a hazard-free dual-rail or 1-hot circuit (depending on the code used). It produces the output only after its both input channels are set to the code word (dual-rail or 1-hot) and resets the output to all-zeroes only after both input buses receive an all-zero separator word. It may have a standard realisation, implementing the function $f(a,b)$: $f = 1 \,|\, a > b \,and\, f = 0 \,|\, a \leq b$, where a and b are input values.

The described delay independent DPA with dual-rail priority buses has been implemented using AMS-0.6μ toolkit and Cadence CAD tools. The simulation results are shown in Fig. 14.

After the design is initialised by applying the *Reset* signal, the request $R0$ is set. The arbiter replies with the grant signal $G0$ almost immediately (4.94ns) as the arc 1 shows. It does not wait for the priority data $P0<0:3>$ to be set, as it is not required to generate the grant. Hazards on the priority bus are prevented by the *done* signal, which indicates the completion of the data setting. The *done* signal may be significantly delayed (as the

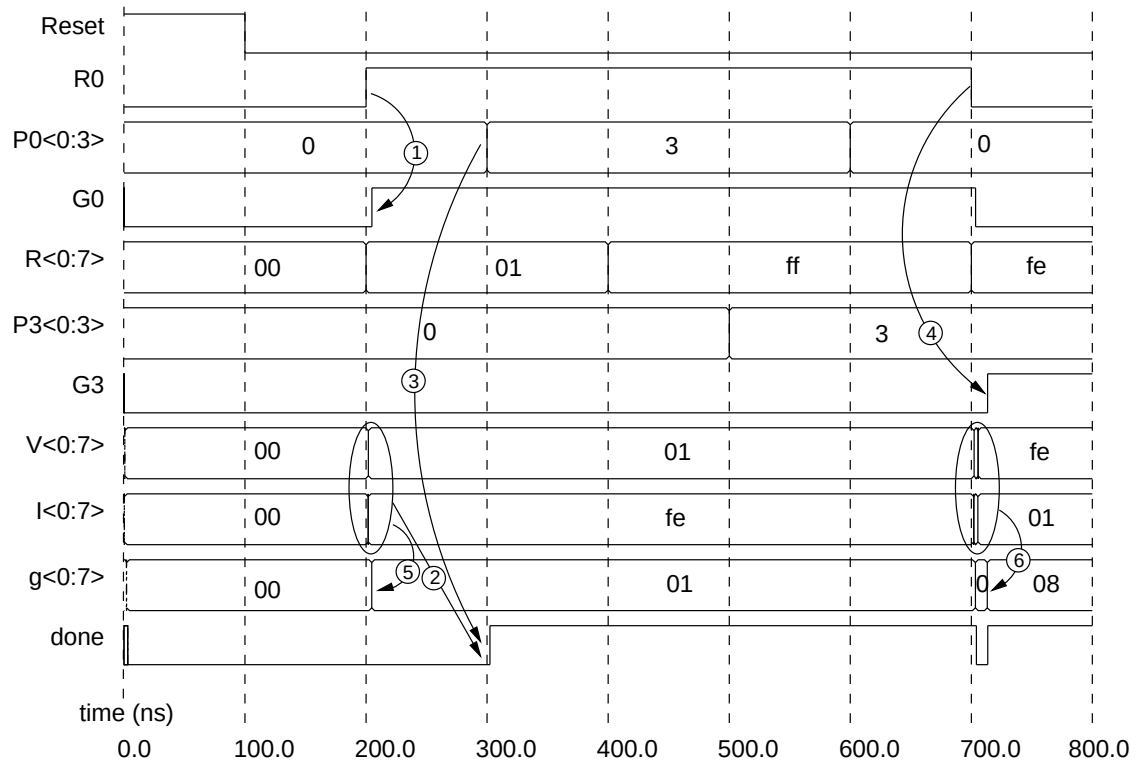


Fig. 14: Waveforms in DPA

arc 2 shows) with respect to V and I signals if data on the priority bus are not ready. This delay, being very long (about 100ns) in the given example, is caused by the external client behaviour and is not related to the arbiter latency. Arc 3 shows the done signal formed after the priority data arrives.

The same figure illustrates pending request processing. Requests $R1 \ldots R7$ ($R<0:7>$ bus) arrive during the the critical section in the client at $R0$ request line. After resetting $R0$, the pending requests $R1 \ldots R7$ begin competing and it takes 13.4ns to generate the grant $G3$ (arc 4). The latency is increased in comparison with $G0$ generation because the priority module cannot compute the output without priority data when multiple requests are pending.

The acceleration technique used in the **PM** 'almost' disregards priority data when the output can be computed without it. Arc 5 shows such an accelerated generation of the primary grant $g0$, which is done in 2.74ns after V and I signals are set. The completion signal *done* is produced much later, when the propagation of data is finished. Arc 6 shows the primary grant calculation when many requests are pending (data comparison is performed in every level of the **PM** tree). This increases the latency to 7.63ns.

The described simulation results show the efficiency of the acceleration technique based on the datapath and control flow decoupling. Further improvement can be achieved by the negative gate optimisation, applying relative timing and using bundled data approach. The latter simplifies the datapath and does not require its resetting, thus making the system faster. The completion of the data manipulation in the bundled data MCCs can

be determined by a programmable delay generator, controlled by the the operating mode logic (similar to that implemented as I18-I21 gates in Fig. 12).

5 Conclusions

We have investigated the problem of Priority Arbitration in the context of a multi-processing system in which client processes gain access to a shared resource on a priority-based discipline. Typical applications for such arbitration schemes are (on-chip) buses and switches.

Conventional techniques for priority arbitration use disciplines which are fixed according to some topological order, such as, e.g., a daisy chain. In our designs we allow the discipline of resource allocation to be a function of parameters of the active requests, which are assigned to the requests either statically or dynamically. We can thus consider systems which are clustered in terms of access to shared resources, and the resource allocation discipline is determined on the basis of a combinatorial function of a vector of active requests and their distribution among clusters. We illustrate this solution in our static priority arbiter (Section 3). The static priority arbiter also provides a generic delay-independent circuitry to lock active requests and issue grants based on a priority module that can be 'plugged-in' or re-programmed with varying delays, without affecting the correctness of the overall logic.

The dynamic case (Section 4) allows priorities to arrive together with their requests, and be involved in the process of computing the grant. In this dynamic case, we design a priority module which is built as a tree with delay-insensitive priority datapath encoding (dual-rail or one-hot). The special feature of this datapath logic is that it uses a three-signal control ('valid'-'invalid'-'done'), which maximally decouples control flow from the data-path by means of an early propagation of the 'valid'-'invalid' signals, concurrently with processing the priority data, separately acknowledged by 'done'. The datapath computation can sometimes be 'disregarded' (depending on the 'valid'-'invalid' inputs) and grant issued with a very low latency. This acceleration significantly reduces the overall arbitration delay when the number of active requests is low.

The design clearly illustrates the power of self-timed design principle in systems where delays are data-dependent. Further speed improvements can be sought in applying relative timing and bundled data techniques, two standard ways of adjusting a generic design to a particular application and process.

We believe that the method with explicit 'valid'-'invalid' control can be extended to other applications involving data processing with asynchronously arriving data from multiple sources. One such application could be embedded microcontrollers.

References

[1] Thorne, N.: 'On-chip buses enable block based ASIC/FPGA design', IP'97-Europe, Oct. 1997, Bracknell.

[2] Clements, A.: 'Microprocessor system design', 3rd edition, Int. Thomson Publishers, 1997.

[3] Del Corso, D., Kirrmann, H., Nicoud, J.D.: 'Microcomputer buses and links', Academic Press, 1986.

[4] Yakovlev, A.: 'Designing arbiters using Petri nets', 1995 Israel Workshop on Asynchronous VLSI, 1995, VLSI Systems Research Center, Israel Institute of Technology, Haifa, Israel, pp. 179-201.

[5] Muller D.E., Bartky: 'A theory of asynchronous circuits', in Proceedings of International Symposium on the Theory of Switching, April 1959, Harvard University Press, pp. 204-243.

[6] Martin, A.J.: 'Collected papers on asynchronous VLSI design', Technical Report, Caltech-CS-TR-90-09, Dept. of Computer Science, California Institute of technology, 1990.

[7] Bystrov, A., Yakovlev, A.: 'Ordered Arbiters', IEE Electronics Letters, 27 May 1999, vol. 35, No 11, pp. 877-879.

[8] Kinniment, D.J., Woods, J.V.: 'Synchronisation and arbitration circuits in digital systems', Proc. IEE, Oct 1976, Vol 123 No 10, pp. 961-966

[9] Josephs MB, Yantchev J. CMOS Design of the Tree Arbiter Element. IEEE Trans. on VLSI Systems, 1996, Vol. 4, No. 4, pp. 472-476.

[10] Yakovlev, A., Petrov, A., Lavagno, L. A Low Latency Asynchronous Arbitration Circuit. IEEE Trans. on VLSI Systems, 1994, vol. 2, No. 3, pp. 372-377