Chapter 4
Arbitration Logic

The kernel of each switch module of the router involves arbiter and multiplexer pairs that need to be carefully co-optimized in order to achieve an overall efficient implementation. Even if the design choices for the multiplexer are practically limited to one or two options, the design space for the arbiter is larger. The arbiter, apart from resolving any conflicting requests for the same resource, it should guarantee that this resource is allocated fairly to the contenders, granting first the input with the highest priority. Therefore, for a fair allocation of resources, we should be able to change dynamically the priority of the arbiter (Synopsys 2009).

The organization of a generic Dynamic Priority Arbiter (DPA) is shown in Fig. 4.1. The DPA consists of two parts; the arbitration logic that decides which request to grant based on the current state of the priorities, and the priority update logic that decides, according to the current grant vector, which inputs to promote. The priority state associated with each input may be one or more bits, depending on the complexity of the priority selection policy. For example, a single priority bit per input suffices for round-robin policy, while for more complex weight-based policies such as first come first served (FCFS), multibit priority quantities are needed.

In this chapter, we will present the design of various dynamic priority arbiters that lead to fast implementations and can implement efficiently a large set of arbitration policies.

4.1 Fixed Priority Arbitration

The simplest form of switch allocators is built using Fixed Priority Arbiters (FPAs), also known as priority encoders. In this case, the priorities of the inputs are statically allocated (no priority state is needed) and only the relative order of the inputs' connections determines the outcome of the arbiter.

G. Dimitrakopoulos et al., *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*, DOI 10.1007/978-1-4614-4301-8_4

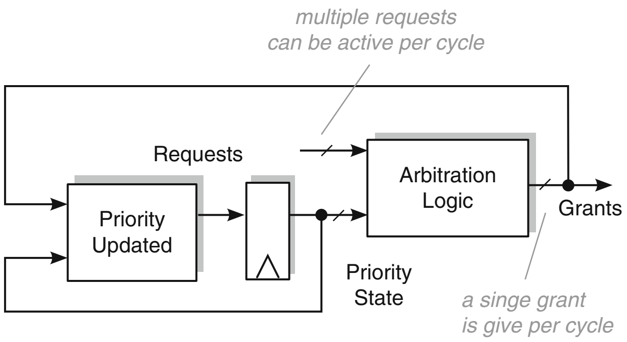


Fig. 4.1 Dynamic priority arbiter allows the priority of each position to change in a programmable way. The selected arbitration policy is implemented in a synergistic way by the arbitration logic and the priority-update logic

In any FPA, the request of position 0 (rightmost) has the highest priority and the request of position $N - 1$ the lowest. For example, when an 8-port FPA receives the request vector $R_7 \ldots R_0 = 01100100$, it would grant input 2 because it has the rightmost active request. When at least one request is granted, an additional flag AG (Any Grant (AG)) is asserted. The FPA can be implemented in many ways. We are interested only in high-speed implementations, where all grant signals are computed in parallel for each bit position (Weste and Harris 2010). In this case, the grant signal G_i is computed via the well-known priority encoding relation $G_i = R_i \cdot \overline{R}_{i-1} \cdot \ldots \cdot \overline{R}_1 \cdot \overline{R}_0$, where $\cdot$ represents the boolean-AND operation and $\overline{R}_i$ denotes the complement of R_i.

An alternative fast implementation can be derived by employing fast adder circuits in the place of priority encoders. At first, we need to derive an intermediate sum by adding 1 to the inverted requests $\overline{R}$. Then, the grant signals are derived via the bitwise AND of the intermediate sum and the original request vector. For example, the complement of the 8-bit request vector $R = 01100100$ is $\overline{R} = 10011011$. Incrementing by one this vector leads to an intermediate sum of 10011100. The bit-wise AND of the sum and the original request vector leads to correct grant vector 00000100 that grants the request of input 2.

Alternatively, fixed priority arbitration can be achieved if we treat the request signals of the FPA as numbers with values 0 and 1, and the fixed priority arbitration as a sorting operation on these numbers (Dimitrakopoulos et al. 2013). Practically, the selection of the rightmost 1, as dictated by the FPA, can be equivalently described as the selection of the maximum number that lies in the rightmost position. Selecting the maximum of a set of numbers can be performed either by a tree or a linear comparison structure. Such structures compare recursively the elements of the set in pairs and the maximum of each pair is propagated closer to the output. Similarly, the sorting-based FPA can be implemented as a binary tree with $N - 1$ comparison nodes. Such a tree, for a 4-port FPA, is shown in Fig. 4.2a. Each node receives two single-bit numbers as input and computes the maximum of the two,

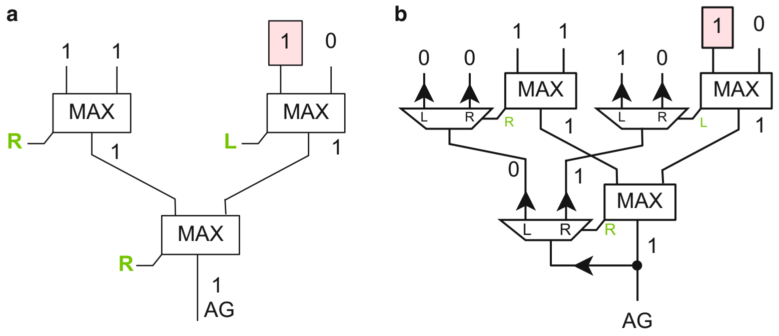


Fig. 4.2 Treating requests as 1-bit numbers transforms fixed-priority arbitration to a maximum selection procedure that identifies the maximum number that lies in the rightmost position. Grant signals are generated according to the values of the direction flags that identify the index of the winning position

along with a flag, that denotes the origin, left or right, of the maximum number. In case of a tie, when equal numbers are compared, the flag always points to the right according to the FPA policy. Note though that when both numbers under comparison are equal to 0 (i.e., between the two compared requests, none is active), the direction flag is actually a don't care value and does not need necessarily to point to the right. In every case, the path that connects the winning input with the output is defined by the direction flags of the MAX nodes.

Each MAX node should identify the maximum of two single-bit input requests, denoted as R_L and R_R, and declare via the direction flag F if the request with the greatest value comes from the left ($F = 1$) or the right ($F = 0$). The first output of the MAX node, that is the maximum, can be computed by the logical OR of R_L and R_R. The other output of the MAX node, flag F, is asserted when the left request R_L is the maximum. Therefore, F should be equal to 1 when $R_L = 1$ and $R_R = 0$.

4.1.1 Generation of the Grant Signals

The maximum-selection tree shown in Fig. 4.2a that replaces the traditional FPA, should be enhanced to facilitate the simultaneous generation of the corresponding grant signals via the flag bits (F). The AG signal at the output of the last MAX is active when at least one grant is generated. Therefore generating grant signals is equivalent to distributing the value of the AG bit to the appropriate input. If the direction of the last node points to the left it means that the value of AG should be propagated to the left subtree and the right subtree should get a zero grant. This distribution is done by the de-multiplexer next to each comparison node shown in Fig. 4.2b. The demultiplexer's input at the root node is the AG bit and its select line is driven by the associated direction flag. When the AG bit propagates to the

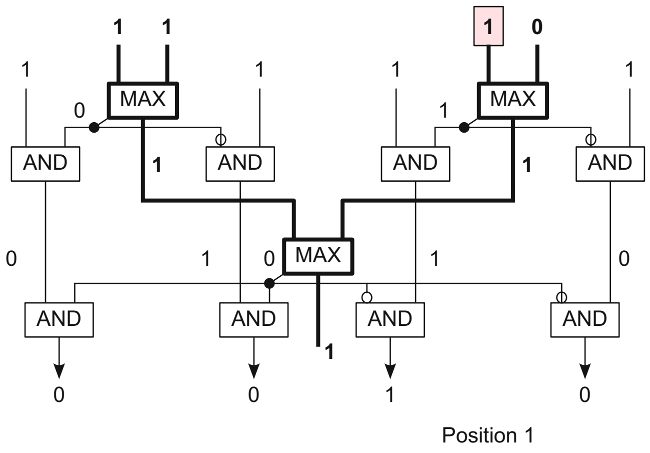


Fig. 4.3 The concurrent computation of the grant signals and the identification of the maximum request

next level of the tree the same operation is performed recursively using a tree of demultiplexers that guide the AG bit of the output to the winning input.

In this way, arbitration and grant generation evolves in two steps. In the first step the maximum rightmost request is identified via the maximum comparison tree and once it is found the winning position is notified by the demultiplexer tree. Instead of waiting the maximum selection tree to finish and then produce the necessary grant signals, the two operations can occur in parallel after the appropriate modification of the grant generation logic that is depicted in Fig. 4.3. In this configuration, all inputs assume speculatively that they will win a grant and thus set their local grant signal to one. In the first level of comparison, each MAX node selects the maximum of the two requests under comparison and its local F flag points to the direction of the winning input. Thus, at this stage, one of the two requests is promoted to the next level of comparison, while the other leaves arbitration. The lost request will never receive a grant. Thus its corresponding grant bit can return to zero. On the contrary, the grant bit of the winning request should be kept active. Keeping and nullifying the grant signals is performed by the AND gates that mask at each level of the tree, the intermediate grant vector of the previous level with the associated direction flags. In the next levels of comparison the same operation is performed. However, when the depth of the tree grows the direction flag should keep or nullify not only two grant bits, but all the grant bits that correspond to the left or right subtree. After the last comparison stage only one grant bit will remain alive pointing to the winning input.

Observe that, if we replace the invert-AND gates of Fig. 4.3 with OR gates, the outcome would be a thermometer-coded grant vector instead of the onehot encoded one.

4.2 Round-Robin Arbitration

Round-robin arbitration logic scans the input requests in a cyclic manner, beginning from the position that has the highest priority, and grants the first active request. For the next arbitration cycle, the priority vector points to the position next to the granted input. In this way, the granted input receives the lowest priority in the next arbitration cycle.

An example of the operation of a round-robin arbiter for 4 consecutive cycles is shown in Fig. 4.4 (the boxes labeled with a letter correspond to the active requests). In the first cycle, input B has the highest priority to receive a grant but does not have an active request. The inputs with an active request are inputs A and C. The arbitration logic scans all requests in a cyclic manner starting from position B. The first active request visited in this cyclic search is input C that is actually granted. In the next cycle, input C should receive the lowest priority according to the round-robin policy. Therefore, priority moves to input D. In this case, the input that is granted is input A since it the first input with an active request when starting searching from input D. Arbitration in the next cycles evolves in a similar manner.

Although there are several approaches for building fast round-robin arbiters like the ones presented in Dimitrakopoulos et al. (2008) and Gupta and McKeown (1999), in this chapter we will describe another solution that leads to equally fast arbiters and its operation is based on a simple algorithmic approach similar to the one presented for FPA (Dimitrakopoulos et al. 2013). A complete overview of all previously presented proposals regarding the logic-level design of round-robin arbiters can be found in Dimitrakopoulos (2010).

The round-robin arbiter utilizes an N-bit priority vector P that follows the thermometer code. As shown in the example of Fig. 4.5, the priority vector splits the input requests in two segments. The high-priority (HP) segment consists of the requests that belong to high priority positions where $P_i = 1$, while the requests,

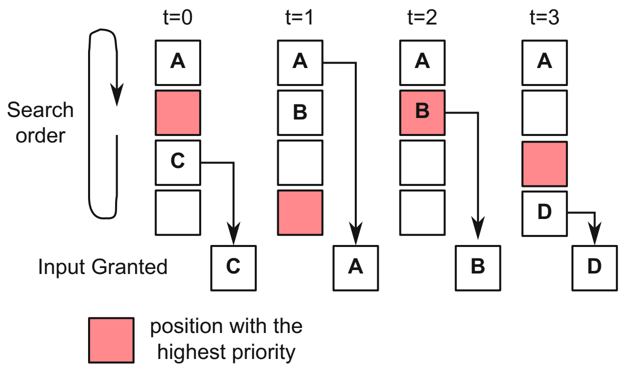
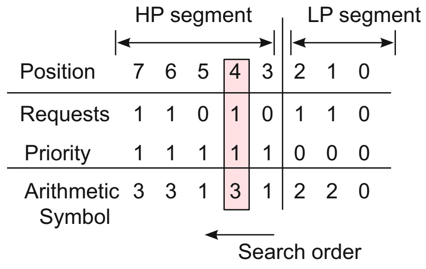


Fig. 4.4 An example of the operation of a round-robin arbiter. The input granted receives the lower priority for the next arbitration round



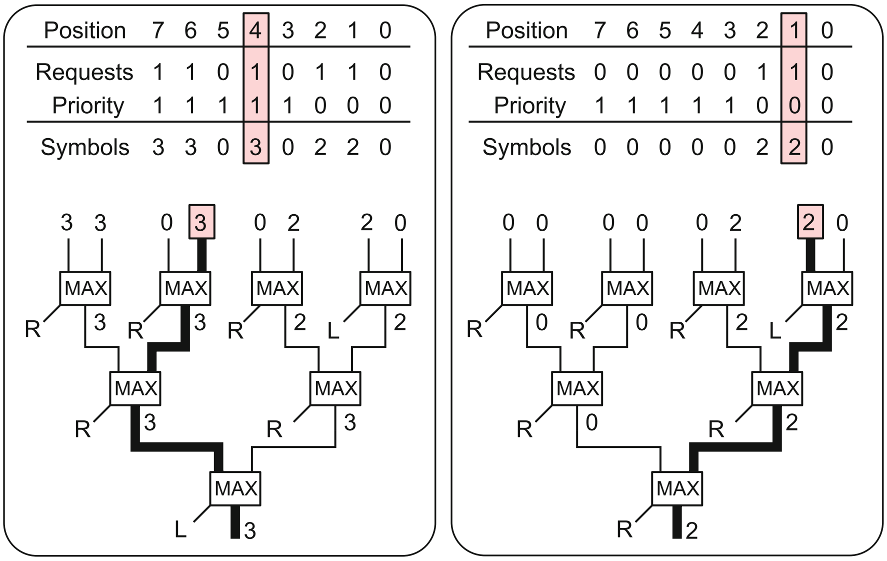
Position	7	6	5	4	3	2	1	0
Requests	1	1	0	1	0	1	1	0
Priority	1	1	1	1	1	0	0	0
Arithmetic Symbol	3	3	1	3	1	2	2	0

Fig. 4.5 The priority vector of the round-robin arbiter separates the requests to a high-priority and a low-priority segment. Treating the requests and the priority bits at each position as independent arithmetic symbols transforms the dynamic cyclic search of round-robin arbitration to an acyclic process that selects the maximum number

which are placed in positions with $P_i = 0$, belong to the low-priority (LP) segment. The operation of the arbiter is to give a grant to the first (rightmost) active request of the HP segment and, if not finding any, to give a grant to the first (rightmost) active request of the LP segment. According to the already known solutions, this operation involves, either implicitly or explicitly, a cyclic search of the requests, starting from the HP segment and continuing to the LP segment.

Either at the HP or the LP segment, the pairs of bits (R_i, P_i) can assume any value. We are interested in giving an arithmetic meaning to these pairs. Therefore, we treat the bits $R_i P_i$ as a 2-bit unsigned quantity with a value equal to $2R_i + P_i$. For example, in the case of an 8-input arbiter, the arithmetic symbols we get for a randomly selected request and priority vector are also shown in Fig. 4.5. From the 4 possible arithmetic symbols, i.e., 3, 2, 1, 0, the symbols that represent an active request are either 3 (from the HP segment) or 2 (from the LP segment). On the contrary, the symbols 1 and 0 denote an inactive request that belongs to the HP and the LP segment, respectively. According to the described arbitration policy and the example priority vector of Fig. 4.5, the arbiter should start looking for an active request from position 3 moving upwards to positions 4, 5, 6, 7 and then to 0, 1, 2 until it finds the first active request. The request that should be granted lies in position 4, which is the first (rightmost) request of the HP segment. Since this request belongs to the HP segment, its corresponding arithmetic symbol is equal to 3. Therefore, granting the first (rightmost) request of the HP segment is equivalent to giving a grant to the first maximum symbol that we find when searching from right to left. This general principle also holds for the case that the HP segment does not contain any active request. Then, all arithmetic symbols of the HP segment would be equal to 1 and any active request of the LP segment would be mapped to a larger number (arithmetic symbol 2).

Therefore, by treating the request and the priority bits as arithmetic symbols, we can transform the round-robin cyclic search to the equivalent operation of selecting the maximum arithmetic symbol that lies in the rightmost position. Searching for the maximum symbol and reporting at the output only its first (rightmost) appearance,



Position	7	6	5	4	3	2	1	0
Requests	1	1	0	1	0	1	1	0
Priority	1	1	1	1	1	0	0	0
Symbols	3	3	0	3	0	2	2	0

Position	7	6	5	4	3	2	1	0
Requests	0	0	0	0	0	1	1	0
Priority	1	1	1	1	1	0	0	0
Symbols	0	0	0	0	0	2	2	0

Fig. 4.6 The round-robin arbiter selects the rightmost maximum symbol. Even if there are no active requests in the HP segment, no cyclic-priority transfer is needed, and the first request of the LP segment is given at the output

implicitly implements the cyclic transfer of the priority from the HP to the LP segment, without requiring any true cycle in the circuit.

The round-robin arbiter that follows this algorithm, is built using a set of small comparison nodes. Each node receives two arithmetic symbols, one coming from the left and one from the right side. The maximum of the two symbols under comparison appears at the output of each node. Also, each node generates one additional control flag that denotes if the left or the right symbol has won, i.e., it was the largest. In case of a tie, when equal symbols are compared, this flag always points to the right. In this way, the first (rightmost) symbol is propagated to the output as dictated by the operation of the arbiter. Two examples of the comparison procedure that implements implicitly the cyclic search of a round-robin arbiter are illustrated in Fig. 4.6.

The associated grant generation logic is exactly the same as the one shown in Fig. 4.3. No change is required since, in both cases, the grants are generated based solely on the local direction flags F and are independent of the operation of the maximum-selection nodes.

4.2.1 *Merging Round-Robin Arbitration with Multiplexing*

In every case, the winning path that connects the winning input with the output is defined by the direction flags of the MAX nodes. Thus, if we use these flags to

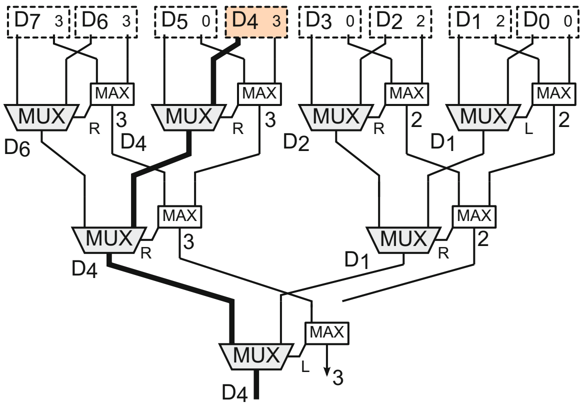


Fig. 4.7 The structure of the merged arbiter and multiplexer implementing the round robin policy

switch the data words that are associated with the input numbers (i.e., the requests), we can route at the output the data word that is associated with the winning request. This combined operation can be implemented by adding a 2-to-1 multiplexer next to each MAX node and connecting the direction flag to the select line of the multiplexer. The organization of this merged round robin arbiter and multiplexer is shown in Fig. 4.7.

4.3 Arbiters with 2D Priority State

Other arbitration policies require the addition of extra priority state bits that treat arbitration and the relative priority of inputs in a different way (Dally and Towles 2004; Satpathy et al. 2012; Boucard and Montperrus 2009). Let's assume that priority is kept in a 2D matrix, where in each position i, j (ith row, jth column) a priority bit is stored that records the relative priority between inputs i and j. When $P[i, j] = 1$, the request from input i has higher priority than the request from input j. To reflect the priority of i over j, the symmetric matrix element $P[j, i]$ should be set equal to 0. Also, the elements of the diagonal $P[i, i]$ have no physical meaning and can be assumed equal to 0. An example priority matrix is shown in Fig. 4.8. In this case, input 0 has a higher priority than input 1 and 2, while input 2 has a higher priority than input 1. By summing the number of ones per row, the total priority order among all inputs is revealed. The input with the largest sum has the highest priority and the rest inputs follow in a decreasing order of sums (priority).

The arbiter receives the current request and the priority matrix and decides which input to grant. Assume at first the case that all requests are active at each arbitration cycle. Then, if at least one 1 exists on column j of the priority matrix then the request of input j cannot be granted, since there is at least one input with higher priority than j. This condition for column j can be identified by ORing all bits of the same column and nullifying the corresponding grant (jth output). However, in

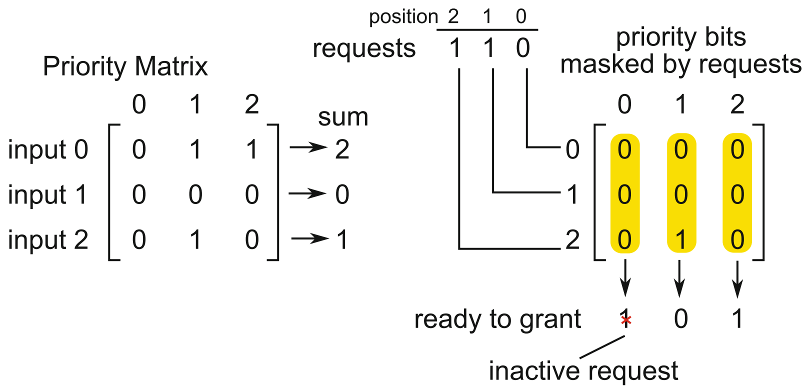


Fig. 4.8 The 2D priority matrix representing the relative priorities of the inputs together with an example of grant generation for on an arbitrary request vector

the general case not all requests are active. Therefore, first the request from each input are ANDed with the priority bits that belong to the same row. Then, from the resulting matrix the inputs that their corresponding column is full of zeroes are eligible to receive a grant (ready to grant). On the contrary, and respecting the relative priority of the inputs, the columns with at least 1 bit asserted should be excluded from the grants.

This operation of the arbiter that relies on a 2D matrix is also shown in Fig. 4.8. Beginning from the priority matrix and masking each row with the corresponding request per input leads to the matrix on the right side of Fig. 4.8. Observing the columns of the matrix we see that column 1 has at least 1 bit asserted which means that another input (input 2 in this case) has a higher priority over input 1. Thus, input 1 cannot receive a grant. On the contrary, inputs 2 and 0 see their corresponding columns filled with zeroes thus they are allowed to receive a grant. By masking the ready to grant bits with the requests gives the final grant vector. Input 0 although had the highest priority and sees a ready to grant bit asserted, it does not receive a grant due to the lack of an active request in the current arbitration cycle. The implementation of the arbitration logic of a 3-input arbiter is shown in Fig. 4.9.

Care should be taken with the priority values since the possibility of a deadlock exists. For example in the case of a 3-input matrix arbiter with $P[0,1] = P[1,2] = P[2,0] = 1$ a circular priority dependency is produced, which blocks the arbiter from producing any grant.

4.3.1 Priority Update Policies

Based on the 2D organization of the priority state, we can derive multiple arbitration policies. The differentiating factor between the possible arbitration policies is on

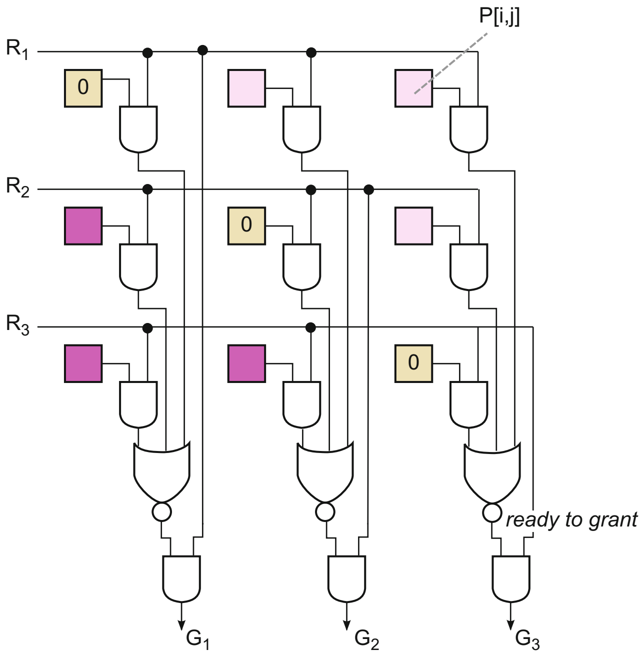


Fig. 4.9 The logic-level implementation of the arbitration logic that receives the input requests and computes the necessary grants based on the relative priorities of the inputs. Priority update logic (not shown in this figure) is designed according to the selected priority update policy

how the priority matrix is updated for the next arbitration round. The arbitration logic remains exactly the same in all cases.

- **Least recently granted:** Once the ith request is granted, its priority is updated and set to be the lowest among all requestors. This is performed at first by clearing all bits of the ith row, e.g., setting $P[i,*]$ to 0, and secondly by setting the bits of the ith column, e.g., $P[*,i] = 1$, so that all other requests will have higher priority over request i.
- **Most recently granted:** Once the ith request is granted, its priority is updated and set to be the highest among all requestors. This is performed at first by setting all bits of the ith row, e.g., setting $P[i,*]$ to 1, and secondly by clearing the bits of the ith column, e.g., $P[*,i] = 0$, so that all other requests will have lower priority over request i.
- **Incremental Round robin:** Under round-robin policy the request that has been granted in the current arbitration cycle should receive the lowest priority in the next cycle. With 1D priority state this is performed in a relatively easy way but becomes very complex in 2D priority representation. Round-robin like operation (or incremental round-robin) proposed in Satpathy et al. (2012) can be achieved by downgrading the position with the highest priority irrespective if it received

a grant or not. The input with the highest priority can be identified by a logical AND operation of all the per-row priority bits. The input that has all the priority bits asserted, i.e., $P[i, *] = 1$, receives the lowest priority for the next arbitration round: $P[i, *] = 0$ and $P[*, i] = 1$. With this update the priority of all other inputs is upgraded by exactly one level.
- **Hybrid First-Come First Served and Least Recently Used:** This priority update policy, proposed in Boucard and Montperrus (2009) tries to combine the benefits of the first-come-first-served and least-recently-used priority-update policies. When a new request arrives at input i and there is no new request at input j, then $P[i, j] = 1$. An input is considered to have a new request, when the request signal changes from 0 to 1 (the detection of this change requires an edge detector with one extra flip-flop for each request line). If the new request from input i is not granted in this cycle, the request is not considered new any more. When, both inputs i and j receive a new request in the same cycle then their relative priority $P[i, j]$ does not change and keeps its old value. At the same time, when the request of an input i is granted in the previous cycle it receives the lowest request in this cycle $P[i, j] = 0$ and $P[j, i] = 1$.

Any other priority update policy can be derived by changing the priority matrix taking possibly into account its current state and the status of the grant signals as depicted in Fig. 4.1.

4.4 Take-Away Points

The arbiter is responsible for resolving any conflicting requests for the same resource and it should guarantee that this resource is allocated fairly to the contenders. The fair allocation of the resources dictates that the arbiter should be able to change dynamically the priority of the inputs and grant in each arbitration round the one with the highest priority. Round-robin arbiters as well as arbiter that store the relative priorities of the inputs in a 2D priority matrix are designed leading to high-speed logic-level implementations.