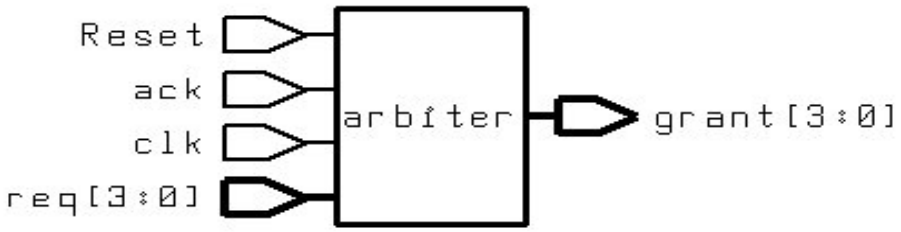
ARBITER

Arbiters are electronic devices that allocate access to shared resources. An arbiter is used whenever we got stuck in a condition where we need to allocate single source to many requesters.

WHY DO WE NEED ARBITER?

Many systems exist in which a large number of requesters must access a common resource. The common resource may be a shared memory, a networking switch fabric, a specialized state machine, or a complex computational element. An arbiter is required to determine how the resource is shared amongst the many requesters.

ROUND ROBIN ARBITER

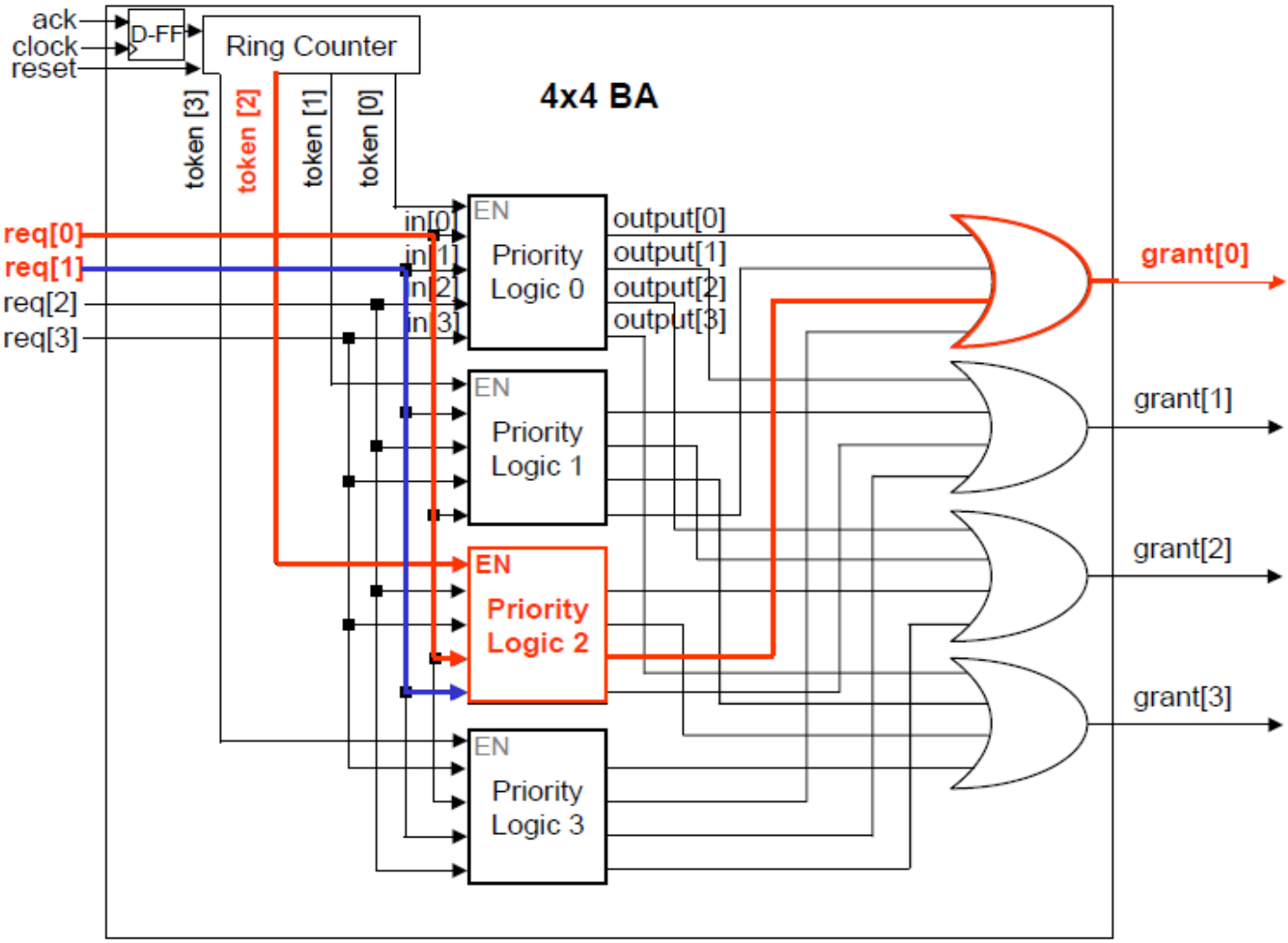


A round-robin token passing bus or switch arbiter guarantees fairness (no starvation) among masters and allows any unused time slot to be allocated to a master whose round-robin turn is later but who is ready now. A reliable prediction of the worst-case wait time is another advantage of the round-robin protocol. The worst-case wait time is proportional to number of requestors minus one. The protocol of a round-robin token passing bus or switch arbiter works as follows. In each cycle, one of the masters (in round-robin order) has the highest priority (i.e., owns the token) for access to a shared resource. If the token-holding master does not need the resource in this cycle, the master with the next highest priority who sends a request can be granted the resource, and the highest priority master then passes the token to the next master in round-robin order.

EN	in [0]	in [1]	in [2]	in [3]	output [0]	output [1]	output [2]	output [3]
0	X	X	X	X	0	0	0	0
1	1	X	X	X	1	0	0	0
1	0	1	X	X	0	1	0	0
1	0	0	1	X	0	0	1	0
1	0	0	0	1	0	0	0	1

Consider a scenario with four processors as bus masters connected to the same bus with one large shared memory on the bus as a slave. Suppose the token is 4 (token=4'b0100, which means processor 2 has the token), and only processor 0 (which uses req[0]) and processor 1 (req[1]) want to access the memory at this cycle. Token=4'b0100 leads to the enabling of only Priority Logic 2 in Figure 3(b). In Priority Logic 2, the connection to in[0] (req[2] from processor 2) indicates the highest priority. Since req[3] is connected to in[1] of

Priority Logic 2 in Figure 3(b), processor 3 has the next highest priority. However, since neither processor 2 nor processor 3 make a request, in[2] which is connected to req[0] is next in line in
priority. Thus, processor 0 is granted access to the memory, and then the memory controller of the accessed memory sends an ack signal, whose connection to the BA is shown in Figure 3(a), indicating when the memory transaction is successfully completed. Next, which could be several processor clock cycles later, the token is passed to processor 3 (the 4-bit ring counter is rotated when the ack signal is received) in which case the token is 4'b1000.


ack
clock
reset
D-FF
Ring Counter
4x4 BA
token [3]
token [2]
token [1]
token [0]
req[0]
req[1]
req[2]
req[3]
in[0]
in[1]
in[2]
in[3]
EN
Priority Logic 0
Priority Logic 1
Priority Logic 2
Priority Logic 3
output[0]
output[1]
output[2]
output[3]
grant[0]
grant[1]
grant[2]
grant[3]