Resource Arbiter

An arbiter is a processor which is required to determine how the resource is shared amongst the clients. It's basically used in a shared memory system to decide, for each memory cycle, which CPU will be allowed to access that shared memory. There are various kinds of arbiters like bus arbiter, round robin arbiter and priority arbiter. Although Arbiter circuits never grant more than one request at a time, there is no way to build an Arbiter that will always reach a decision within a fixed time limit.

Present-day Arbiters reach decisions very quickly on average, usually within about a few hundred picoseconds. When faced with close calls, however, the circuits may occasionally take twice as long, and in very rare cases the time needed to make a decision may be 10 times as long as normal

The controller accepts requests from three clients **A (01), B (10), C (11)**. A client requests access to a resource (memory location) when he wakes up from state of sleep. This is the request signal and the client generates two four bit distinct random.

The client basically requests the controller to block access of the resource for r1 time units, and if successful, he is notified and sleeps for r2 time units else he is kept waiting. If the resource is blocked for a client, requests from other clients are kept in a waiting line in the same order and the first one is serviced as soon as the block is over.

Arbiter Function:

It contains two modules- ranNUM and arbiter.

ranNUM:

The random numbers are generated in the background in the ranNUM module by using {urandom}%n function such that 0<r1<r2<16 and when request is high they are stored into corresponding variables for client one, two and three into r1_1, r1_2 for A , r2_1, r2_2 for B and r3_1 ,r3_2 for C.

Queue:

In the **arbiter module**, wake signals are taken as a wire input and the value is assigned to reg awake and when it is 1, it is requesting access (req=1). Let 01, 10 and 11 denote client A, client B and client C respectively. Consider the 6bit register, **queue**, which stores the order of the waiting line. Eg: if A has blocked

access then queue[0]=1 and queue[1]=0 and if B requests access, queue[2]=0 and queue[3]=1. After r2 cycles, A is popped out and queue[0]=0 and queue[1]=1, i.e. B is granted access when we shift right by 2 bits. The indices of queue are kept track of by using i.

If the resource has to be blocked for a particular client, say A(01), then when attendA changes to high and queue[1] and queue[0] are corresponding to 01, then grantA is high and the memory is blocked for A.

Output:
The output is the client who has access to the resource/memory location currently. It is the unique 2bit number that we have assigned for each client.

Eg: if A has blocked the memory location, then the output will 01.

BLOCK DIAGRAM: