BUS ARBITER:

It is desired that more than one independent processor in system further requirement is that they require access to same set of system resources for ex memory size is 64 KB and require a single RD/WR signal.

Design a system that accept data from each independent processor and arbitrate which one is granted access to memory at any one time.

Each independent processor will initiate a memory-required signal when it wants access to memory and will deactivate the same when the job is over. if more than one processor request for the bus at the same time , access should be granted on round robin basis.

This is in order to ensure that no one independent processor is locked out while another has continuous access. Continuous access is to be granted to any one processor for a period of time. This time period should be separately programmable from the data bus of one the period is not explicitly set, a 128 clock cycle delays should default.

The design should aim at speed efficiency.

SOLUTION TO THE PROJECT:

1. ASSUMPTIONS TO THE BUS ARBITER:

1) There are three independent processors say A, B, C.
2) It is assumed that the priority in the order A> B> C.
3) Duration of access time (Timeout period) is programmed through the independent processor and the data bus.

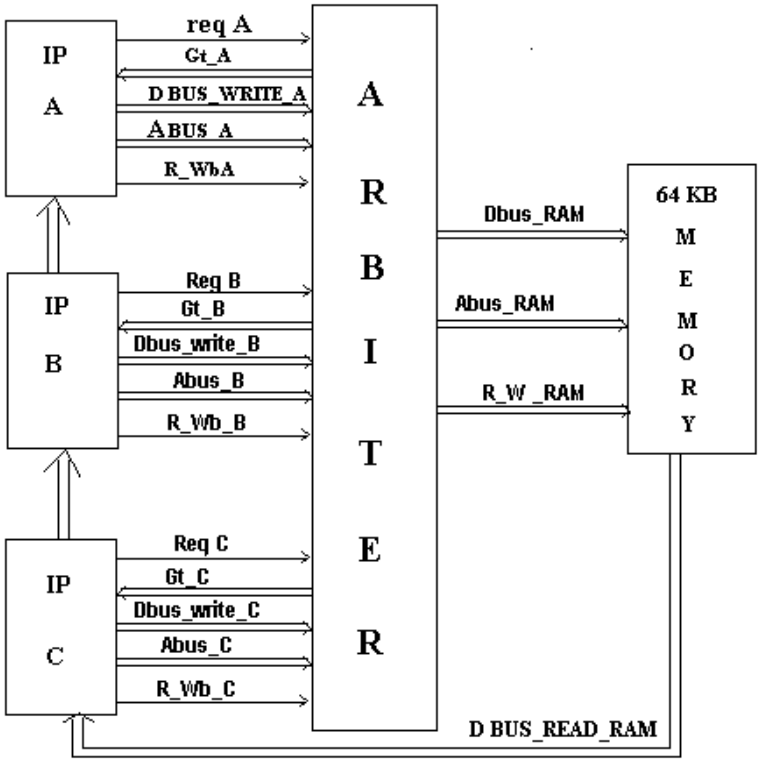
2. OBSERVATION FROM SPECIFICATIONS:

1) Memory size = 64 KB this implies 16-bit address bus and 8-bit data bus.
2) Default time out period= 128 clock cycle.
3) Multiplex read/write pin.
4) Bus structure should employ tri state logic.

3. BASIC BLOCK SCHEMATIC:

This data is quite sufficient information to arrive at a block level schematic of the overall project as specified by the project specifications. System should include:

1) Three independent requesting processors
2) An arbiter, which will resolve the shared bus conflict.
3) Memory of size 64 KB.



4. SEQUENCE OF EVENTS:

1) All the three processor have request access to bus either for reading or writing of data from/or to the memory.
2) Since priority of IPA is highest, the arbiter will grant access of buses to A and will deny access of buses to processor B and C as well.
3) The processor A in turn will specify the type of memory reference is memory read or memory write. Using this information arbiter will issue address as provided by processor and respective control signal for memory and the operation is continued.
4) Being in this state, arbiter also keep track of the duration of access (Timeout period) and take appropriate action.

10. OBSERVATION FROM SEQUENCE OF EVENTS:

1. Arbiter at any instant of time is in one of the following states:

    a) Serving A
    b) Serving B
    c) Serving C
    d) Sitting idle.

2. During the time when it is serving any of independent processor it is keeping track of time out period and/or the priorities and the request from remaining IPs.

3. Being idle, remains idle until receives request from one or more independent processor.

11. CLOSE TO IMPLEMENTTATION:

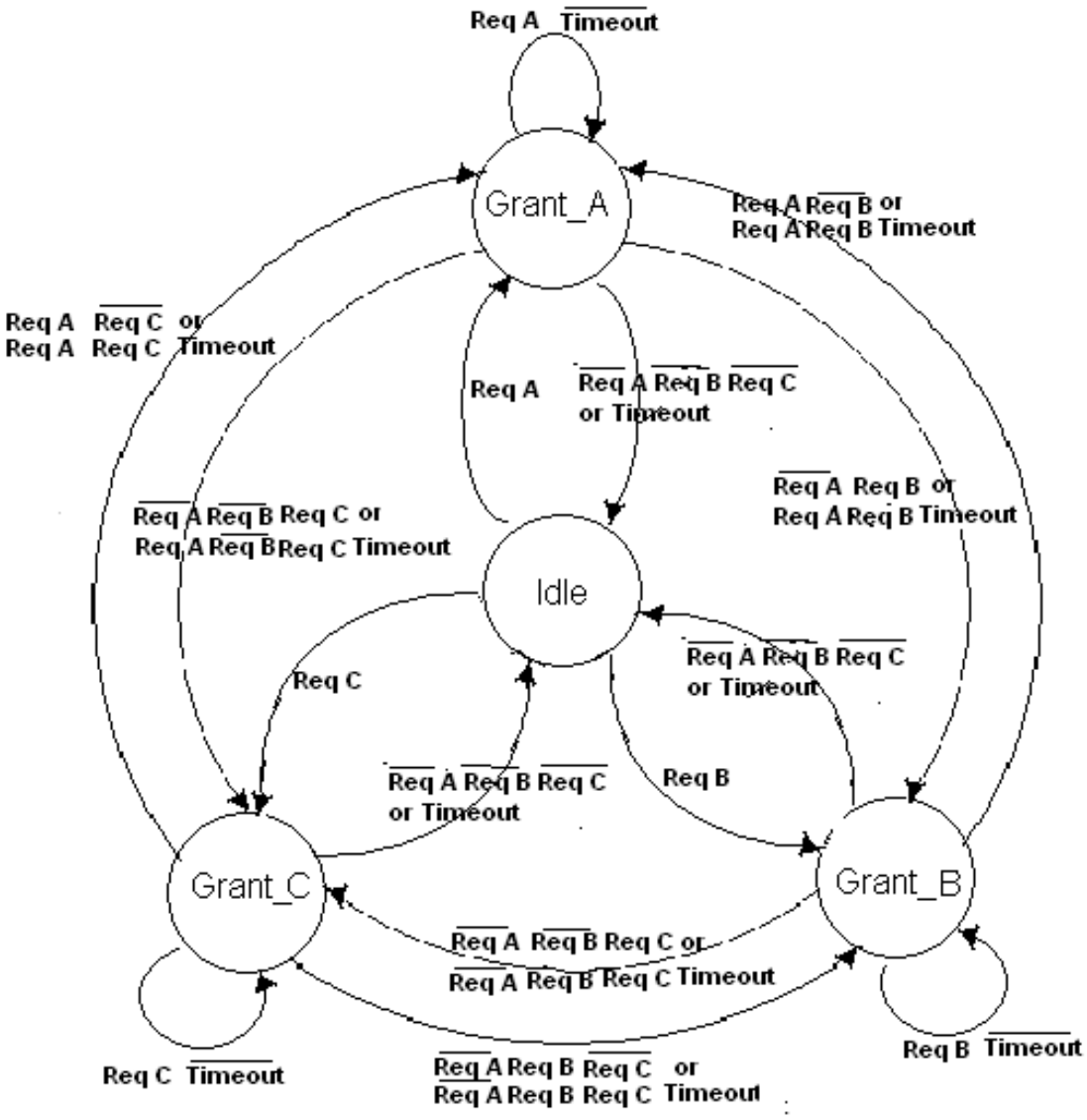
At this point of time we have a fair amount of idea as how to implement an arbiter and what modules we would require to implement the same.

1) A watchdog timer would be needed to keep track of time out period of respective processors.
2) The state machine having four states says idle, ServeA, ServeB and ServeC.
3) Finally the multiplexer logic or tri-state logic for routing information.

12. STATE DIAGRAM:

As we has already been pointed out an arbiter can find itself in one of the four states is an idle, ServeA, ServeB or ServeC. Now it's time to design a state machine is to identify the possible input to state machine, formulate next state logic depending upon input and current state and finally to come up with the output logic.

Let us initially assume that the current state is idle state being in this state the next state is governed by the request inputs only. The possible combinations are as shown.



If the arbiter is in the state ServeA or ServeB or ServeC the, determination of next state is critical and is governed not only by complex combination of input ReqA ReqB , ReqC and Time Out . The input time out here ensures that the bus does not remain locked with a particular processor for duration greater than the time out period.

OUTPUT LOGIC:

COMBINATIONAL OUTPUTS:

Current State	Input	Next state	Outputs
Idle	Reg A	Grant_A	Gt_A
Idle	Reg B	Grant_B	Gt_B
Idle	Reg C	Grant_C	Gt_C
Idle	$\overline{\text{Reg A}}$ $\overline{\text{Reg B}}$ $\overline{\text{Reg C}}$	Idle	$\overline{\text{Gt\_A}}$ $\overline{\text{Gt\_B}}$ $\overline{\text{Gt\_C}}$
Grant_A	Reg A, $\overline{\text{Time out}}$	Grant_A	Gt_A ,Count_down
Grant_A	Reg B	Grant_B	-
Grant_A	Reg C	Grant_C	-
Grant_A	Reg A Reg B Reg C Time out	Idle	-

NOTE: This is prioritized logic with IP 'A' having higest priority & 'C' having lowest priority. Thus simultaneous request would lead to access of bus to the IP with higher priority at any instant of time.

Current State	Input	Next state	Outputs
Grant_B	Req B, Time out bar	Grant_A	Gt_A ,Count_down
Grant_B	Req A	Grant_A	-
Grant_B	Req C	Grant_A	-
Grant_B	Req A bar ,Req B bar, Req C bar or Time out	Idle	-
Grant_C	Req C, Time out bar	Grant_A	Gt_C ,Count_down
Grant_C	Req A	Grant_A	-
Grant_C	Req B	Grant_A	-
Grant_C	Req A bar ,Req B bar, Req C bar or Time out	Idle	-

SEQUENTIAL OUTPUT:

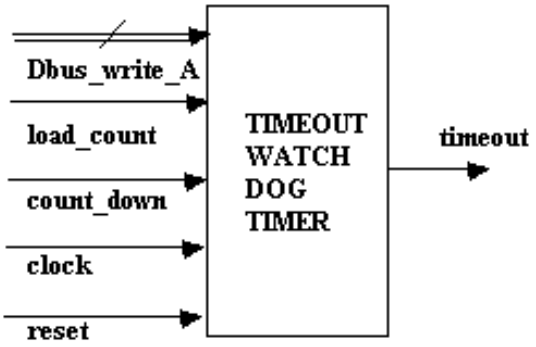
This is the time where we must look into the future in order to grant the bus access to a particular processor depending upon the current state of arbiter. Thus we need to generate some kind of enable signal so as to enable the bus logic of concerning independent processor.

As we know that we have three sets of buses, address, data and direction control (read/write bar) we need to enable all three set of buses, for this purpose we will generate two enable signal {one for address bus and second for data and control bus} so as to ensure reduced loading on a single driver.

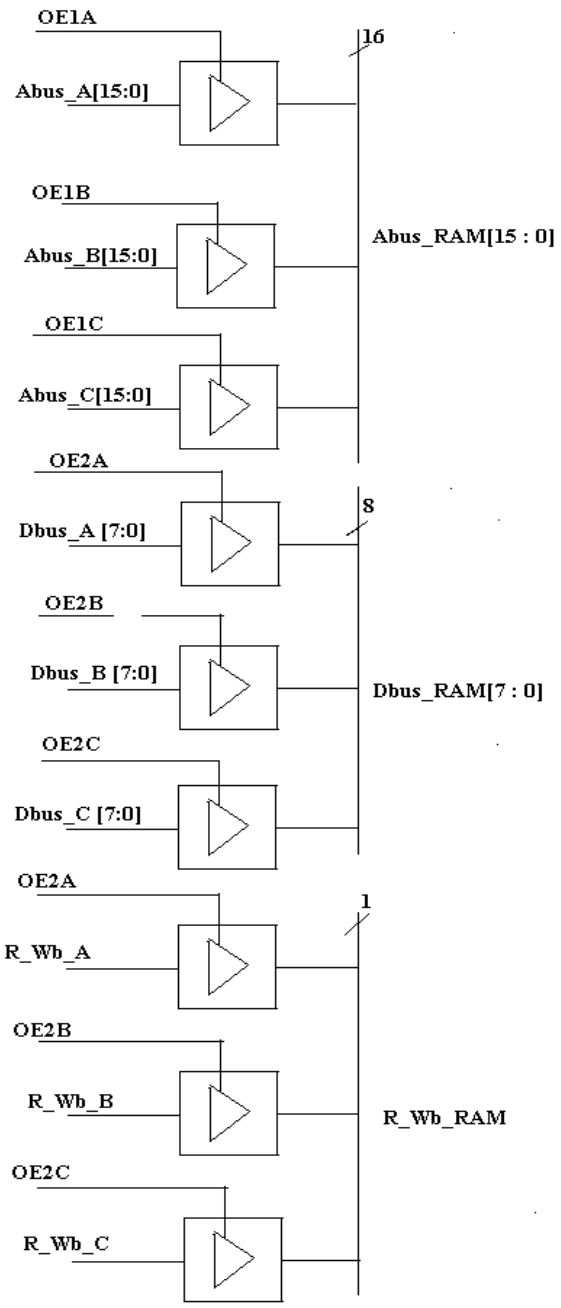
This is tabulated as under:

Current State	Outputs
Grant_A	OE1A OE2A $\overline{OE1B}$ $\overline{OE2B}$ $\overline{OE1C}$ $\overline{OE2C}$
Grant_B	$\overline{OE1A}$ $\overline{OE2A}$ $\overline{OE2C}$ OE1B OE2B $\overline{OE1C}$
Grant_C	$\overline{OE1A}$ $\overline{OE2A}$ $\overline{OE1B}$: $\overline{OE2B}$ OE1C, OE2C
Idle	$\overline{OE1A}$ $\overline{OE2A}$ $\overline{OE1B}$ $\overline{OE2B}$ $\overline{OE1C}$ $\overline{OE2C}$

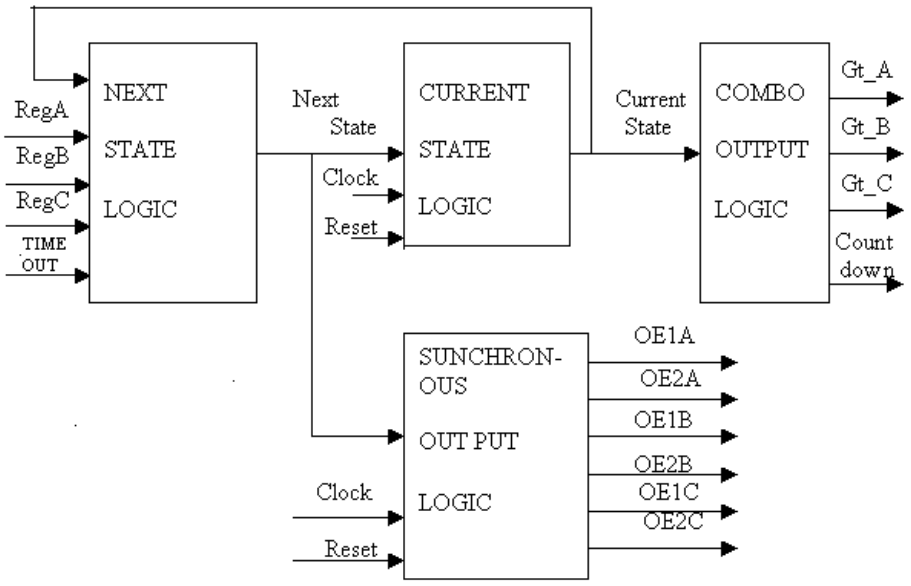
INTERNAL MODULES OF BUS ARBITER:



BLOCK DIA OF WATCH DOG TIMER



TRISTATE BUS STRUCTUREOF BUS ARBITER OUTPUT



THE COMPLETE STATE MACHINE

BUILT-IN SELF TEST:

As digital systems become more and more complex, they become much harder and more expensive to test. One solution to this problem is to add logic to the IC so that it can test itself. This is referred to as Built-in Self-test, or BIST. Following figure illustrates the general method for using BIST. When the test mode is selected by the test signal, an on-chip test generator applies test patterns to the circuit under test. The resulting output is observed by the response monitor, which produces an error signal if an incorrect output pattern is detected.