A NEW ARBITRATION CIRCUIT FOR SYNCHRONOUS MULTIPLE BUS MULTIPROCESSOR SYSTEMS

Syed Masud Mahmud and Md. Showkat-U-Alam
Electrical and Computer Engineering Department
Wayne State University, Detroit, MI 48202
Phone: (313) 577-3855
mahmud%wayne-mits@um.cc.umich.edu

Abstract: A multiple bus structure is very attractive for interconnecting the processors and memory modules of a multiprocessor system. A bus arbitration circuit is necessary for assigning the busses to the different devices for transferring information. In this paper we have presented a new design for an M-user B-server arbiter for a multiple bus system. Our arbitration circuit maintains fairness when it is used in a low-order interleaved memory system. Our design is also faster than the previous published work.

I. INTRODUCTION

In a shared memory multiprocessor system, two or more processors may simultaneously request the use of a shared resource. Therefore, an arbitration circuit is essential to resolve the contention among the competing processors and allocate the resource to the appropriate requesting processor (according to an arbitration protocol). A number of arbiter designs for multiprocessor systems have been reported in the literature [1-9], but the designs presented in [1-7] are suitable for single bus multiprocessor systems. An one-step programmable arbiter is proposed by K. Holberg [1], in which only one state transition is required from a free to an allocated resource with arbiter structure and status programmed into state machine memory. A mufiarequest and uniserver arbiter has been presented in [5] which adopted an asynchronous, centralized scheme. For a multiprocessor system with large number of processors and memory modules, a centralized bus arbiter may become a serious bottleneck since all bus requests must be handled by one arbiter which will seriously affect the performance and reliability of the system. The most widely accepted arbiter design for single bus asynchronous multiprocessor systems has been invented by D.M. Taub [5] and is implemented in IEEE 896 futurebus. For a detailed analysis of arbitration protocols and their implementation for single bus multiprocessor systems, the reader is referred to reference [7]. References [8-9] addressed the arbitration mechanism for multiple bus multiprocessor systems. A synchronous M-user B-server arbiter is proposed for a multiple bus system by T. Lang and M. Valero [9]. In this architecture, a lookahead technique is used to reduce the delay of arbitration process. The authors presented three different designs, viz., iterative, I-level lookahead, and 2-level lookahead for this arbiter. The iterative design (also referred in [8]) involves the lowest cost and the highest delay while the 2-level lookahead involves the highest cost and the lowest delay. This technique may be employed to build an arbiter for multiple bus multiprocessor systems. But the delay and cost associated with this approach will be excessive. In this paper, we propose a new design for an M-user B-server arbiter. Our design is less complex and faster than the design presented in [9].

II. OPERATION OF THE PROPOSED ARBITER

A multiple bus multiprocessor system with N processors, M memory modules, and B buses generally requires two types of arbiters: M number of N-to-1 arbiters to select among N request inputs each associated with a processor and one M-to-B arbiter to assign buses to the memory modules which have been successfully accessed by the requesting processors. An N-to-1 arbiter design consists of a binary tree of 2-to-1 arbiters. The circuit diagram of our modified 2-to-1 arbiter is shown in Figure 2b which has two request lines R0 and R1, two grant lines G0 and G1, a cascaded request output Rc, and a cascaded grant input Gc. If a request is made by raising an input
constructed as a multistage arbitration network, where a bus number. Each bkdc has additional logic gates to handle the grant lines and granted signal. The highest stage is built using Type-3 blocks. The description of the Type-1, Type-2, and Type-3 arbiter blocks are given below. The block diagrams of the Type-1, Type-2 and Type-3 arbiter blocks with input and output signals are shown in Figures 3b and 3c.

Type-1 Block: The circuit diagram of this block is shown in Figure 4, which consists of an state flip-flop plus additional logic gates to handle the grant lines and granted bus number. Each block has two request input lines \( R_0 \) and \( R_1 \), two grant input lines: a primary grant input line \( G_0 \) and a secondary grant input line \( G_2 \) from the next higher stage of the network. There are four main output lines from each block: a primary request line \( R_0 \) (higher priority - indicated by a long arrow), a secondary request line \( R_2 \) (lower priority - indicated by a short arrow), and two grant lines: \( G_0 \) and \( G_1 \). In addition to the above mentioned input/output lines, there are two input \( (B_0 \) and \( B_1) \) and two output \( (B_2 \) and \( B_3) \) ports to transmit the bus numbers received from the last higher stage to the next lower stage. If a request for a shared resource is made by raising only one of the two lines: \( R_0 \) and \( R_1 \), then this request is always transmitted to the next higher stage through the higher priority line \( R_0 \). In case of two simultaneous requests (i.e., \( R_0=1 \) and \( R_1=1 \)), the transmission of the request signals to the next higher stage will depend on the status (\( Q \)) of the state flip-flop. If both \( R_0 \) and \( R_1 \) are high and \( Q \) is reset, then \( R_0 \) will get priority over \( R_1 \), that means \( R_0 \) will go through the output line \( G_0 \), and \( R_1 \) will go through the output line \( G_1 \). But if \( Q \) is set, then \( R_1 \) will get priority over \( R_0 \). If a request propagates through the line \( R_0 \) then the corresponding grant output will be equal to the grant input \( G_0 \). If a request is low then its grant will be a don't care term. For example, if \( R_0 \) goes through the output line \( G_0 \) then \( G_0 \) will be equal to \( G_0 \). Similarly, if a request propagates through the line \( R_2 \) then the corresponding grant output will be equal to the grant input \( G_2 \). If a request is low then its grant will be a don't care term. For example, if \( R_0 \) is equal to 0 then the grant signal \( G_0 \) would be a don't care term, that means any value could be assigned to \( G_0 \). This don't care value will not create any problem in the operation of the bus arbitration circuit. Because, if a request line is low then it means that no device has requested to share that particular memory module. Thus, no device will be monitoring that don't care grant output line.
The status of the state flip-flop is updated at the end of the arbitration cycle. If both the input requests \( R_0 \) and \( R_1 \) are low (i.e., no request) then the state of the flip-flop does not change. If only \( R_1 \) is high then \( Q \) will be reset if the request gets service. If only \( R_0 \) is high then \( Q \) will be set if the request gets service. If only one request is made and if it does not get the service then the state of the flip-flop is not changed. If both \( R_0 \) and \( R_1 \) are high then the state of the flip-flop will not change if the higher priority request does not get the service (i.e., if \( G_p=0 \)). Otherwise, the state of the flip-flop will toggle.

The Boolean equations of the different input/output signals and the next state \((Q^+)^{\prime}\) of the state flip-flop are as follows:

\[
R_p = R_0 + R_1
\]

\[
R_s = R_0 \overline{R_1}
\]

\[
Q^+ = R_1 G_p \overline{Q} + \overline{R_1} G_p Q
\]

\[
G_o = \overline{R_0} G_p + \overline{R_0} \overline{G_o} + R_0 G_s
\]

\[
G_1 = \overline{R_0} G_p + \overline{R_0} G_s
\]

When a request is made and a grant signal is available, the number (in binary) of the available bus received from the last higher stage through the input bus port bits will be transmitted through the output bus port bits to the next lower stage. The Boolean equation of a bit, say bit-\( i \), of the primary output port is the same as that of the grant line \( G_0 \) with \( G_p \) and \( G_s \) replaced by \( B_p(i) \) and \( B_s(i) \) respectively. Where, \( B_p(i) \) and \( B_s(i) \) are the \( i \)th bit of the primary and secondary input ports respectively. Similarly, the Boolean equation of bit-\( i \) of the secondary output port is the same as that of the grant line \( G_1 \) with \( B_p(i) \) and \( B_s(i) \) replaced by \( B_p(i) \) and \( B_s(i) \) respectively.

Type-2 Block: The basic difference between a Type-1 block and a Type-2 block is that a Type-2 block contains only combinational circuits as shown in Figure 5a. The input and output lines of this block are the same as that of a Type-1 block. In a Type-2 block, request \( R_0 \) always has the higher priority than request \( R_1 \). That means, if both \( R_0 \) and \( R_1 \) are high then \( R_0 \) will go through the output line \( R_p \) and \( R_1 \) will go through the output line \( R_s \). But, if a request is made only through one of the two lines: \( R_0 \) and \( R_1 \), then that request always goes to the next higher stage through the higher priority line \( R_p \). If a request goes through the output line \( R_0 \) then grant signal of that request is equal to \( G_0 \), otherwise the grant signal is equal to \( G_1 \). The Boolean equations of the output lines \( R_0 \) and \( R_1 \) of this block are the same as those of a Type-1 block, while the equations of the lines \( G_0 \) and \( G_1 \) are given below:

\[
G_0 = G_p
\]

\[
G_1 = \overline{R_0} G_p + R_0 G_s
\]
The Boolean equations of the bus port bits can be derived from the above equations following the procedure described for the Type-1 block.

Type-3 Block: A Type-3 block is similar to that of a Type-2 block except that it contains one request output line, one grant input line and one input port to transmit the available bus number. The governing equations of this block are similar to those of a Type-2 block with $G_S$ and $B_S(i)$, for all $i$, equal to zero. The circuit diagram of this block is shown in Figure 5b.

The design of M-to-B arbiters for different values of $M$ and $B$ are shown in Figure 6. If both $M$ and $B$ are powers of 2 then the total number of stages, $S$, in an M-to-B arbiter can be expressed as:

$$S = B \log_2 (M/B)$$

If $B$ is a power of 2 but $M$ is not a power of 2 then the upper bound on the total number of stages in an M-to-B arbiter can be expressed as:

$$S = B \left \lfloor \log_2 (M/B) \right \rfloor$$

A practical implementation of an M-to-B arbiter can easily be made with existing VLSI technology. For example, an 8-to-4 arbiter has 12 request, 12 grant, 24 bus number I/O lines and one clock input line. Considering one pin for power supply and another pin for ground, we observe that this arbiter module can be built by fabricating the proposed logic circuits in an integrated circuit (IC) with 50 pins. Similarly other single chip arbiter modules can be implemented.

III. PERFORMANCE OF THE ARBITER

In this section, the performance of our M-to-B arbiter is shown in terms of the cost, speed and fairness. The performance of our arbiter is also compared with the arbiter presented in [9].

Cost:
If the number of buses, $B$, is a power of 2 (say, $B=2^b$), then a Type-1 block (refer to Fig. 4) of the proposed arbiter contains $2^{b+6b}$ gates, a Type-2 block contains $6+3b$ gates, and a Type-3 block contains $3+b$ gates. In this design the maximum fan-in of the gates is 3. Thus, if the optimization goal is to reduce the number of gates, then an 8-to-4 arbiter can be built using only $(3*4 + 12*8 + 5*4) = 248$ gates. For higher number of memory modules, the gate count is shown in Table-1.

Speed:
Assuming $\delta$ to be the nominal gate delay, for any type of arbiter block, the delay from the request inputs to the request outputs is $\delta$. For Type-1 and Type-2 arbiter blocks, the delay from the grant inputs to the grant outputs is $2\delta$, for a Type-3 arbiter block, the delay from the grant input to the grant outputs is $\delta$. The time required to update the states of the flip-flops is not considered as a part of the arbitration delay, because the flip-flop states are updated when information is transferred through the buses. Since all the flip-flop states are updated simultaneously using the synchronous clock, we can assume that the time required to update the states of the flip-flops is very small as compared to the time required for transferring information. Thus, this state-update time has no effect on the bus arbitration time. An 8-to-2 arbiter has four stages, as shown in Figure 7. Two of these stages are built by Type-1 blocks and other two stages are built by Type-3 blocks. Therefore, the maximum delay for an 8-to-2 arbiter is $10\delta$. Similarly it can be shown that the maximum delay for 8-to-4 and 16-to-4 arbiters are $11\delta$.
and 226 respectively. If both M and B of an M-to-B arbiter are powers of 2, then the general expression for the delay would be

\[ D = (\log_2(M/B)) \times (3B - 116) \]  

(10)

However, the value of delay could be reduced without increasing the gate count significantly, if the output requests of an M-to-B arbiter are directly implemented in a 2-level logic as a function of the input requests. For example, if this technique is used to reduce the delay for an 8-to-4 arbiter then the total gate count increases from 248 to 252 (1.6% increase in cost), and the delay is reduced from 116 to 96 (18.2% decrease in delay). Further reduction in delay is possible if all the combinational logic circuits are implemented in fewer levels. An 8-to-4 arbiter can be implemented using only 268 gates and the total delay can be reduced to 55 if the output requests and output grant signals are implemented using a maximum of 3-level logic. But, this implementation will require some gates with a maximum fan-in equal to 8. This high speed 8-to-4 arbiter can then be used as a basic arbiter module to build any M-to-4 arbiter, and if M is a power of 2 then the delay for an M-to-4 arbiter can be expressed as

\[ D = 56 \times (\log_2(M/4)) \]  

(11)

If only 2-level logic circuits are used then the total gate count for an 8-to-4 arbiter will be 376, and the total delay will be 48, where a delay of 25 is for generating output requests and another delay of 23 is for generating output grant signals. As a result, equation (11) can be rewritten as

\[ D = 46 \times (\log_2(M/4)) \]  

(12)

for an M-to-4 arbiter with M equals to a power of 2. The gate count and the delay of M-to-4 arbiters, built using the 8-to-4 arbiter modules with minimum delay, are shown in Table-2 for different values of M.

**Fairness:**

Fairness is maintained in our proposed M-to-B arbiter due to the presence of the state flip-flop in the

During every memory cycle memory requests were generated based on the value of \( p_0 \). In our simulation it was assumed that the memory requests for different memory modules are independent. All blocked requests in any memory cycle were resubmitted during the next memory cycle, and new memory requests were generated for other memory modules based on the value of \( p_0 \). The simulation results show the mean number of trials needed to get a bus. If a memory request gets service at the first trial, then we say that the number of trials required for this request to get a bus is 1. Column 2 of Table-3 shows the overall mean, and columns 3 through 10 show the mean trials required for all the memory modules. From the simulation results it is seen that the mean trials necessary for all the memory modules are almost the same. Thus, we can claim that our 8-to-2 arbiter is fair, and also no memory request will hang up for an indefinite period of time. It can easily be proved that for any M-to-B arbiter the maximum number of trials needed to get the bus would be equal to \( M/B \) if both M and B are powers of 2.

<table>
<thead>
<tr>
<th>Type of arbiter</th>
<th>No. of Type-2 blocks req.</th>
<th>No. of Type-3 blocks req.</th>
<th>No. of Type-4 blocks req.</th>
<th>No. of Type-5 blocks req.</th>
<th>No. of Type-6 blocks req.</th>
<th>Total no. of gates</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-to-2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>376</td>
<td>46</td>
</tr>
<tr>
<td>16-to-4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1128</td>
<td>85</td>
</tr>
<tr>
<td>32-to-4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>2632</td>
<td>120</td>
</tr>
<tr>
<td>64-to-8</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>5640</td>
<td>160</td>
</tr>
</tbody>
</table>

**Table-1**: Gate counts and block counts for different types of M-to-4 arbiters (Minimum Gate Logic)

**Table-2**: Gate counts and block counts for different types of M-to-2 arbiters (Minimum Gate Logic)

Since Lang and Valero's work [9] has been referred by most of the authors who have published papers in the area of bus arbitration, we compared the performance of our arbiter with the performance of Lang and Valero's M-user B-server arbiter. We have computed the number of gates required to implement their arbiter. We have found that if more than 4-input gates are not used in the implementation, then Lang and Valero's design with 2-level lookahead requires approximately 1900 gates for a 16-user 4-server arbiter, and the arbitration delay is approximately 506. Where, \( \delta \) is the typical delay of a gate. For the same arbiter, our design requires only 744 gates with an arbitration delay of 226, as shown in Table-1.
Table 3: Average number of trials and the maximum number of trials (in worst case) required to get a bus

<table>
<thead>
<tr>
<th>( p_0 )</th>
<th>Average number of trials required to get a bus</th>
<th>Trials reqd in worst case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>0.05</td>
<td>1.037</td>
<td>1.04</td>
</tr>
<tr>
<td>0.10</td>
<td>1.100</td>
<td>1.11</td>
</tr>
<tr>
<td>0.15</td>
<td>1.197</td>
<td>1.20</td>
</tr>
<tr>
<td>0.20</td>
<td>1.363</td>
<td>1.38</td>
</tr>
<tr>
<td>0.25</td>
<td>1.625</td>
<td>1.63</td>
</tr>
<tr>
<td>0.30</td>
<td>1.904</td>
<td>1.90</td>
</tr>
<tr>
<td>0.35</td>
<td>2.240</td>
<td>2.24</td>
</tr>
<tr>
<td>0.40</td>
<td>2.524</td>
<td>2.52</td>
</tr>
<tr>
<td>0.45</td>
<td>2.784</td>
<td>2.77</td>
</tr>
<tr>
<td>0.50</td>
<td>2.994</td>
<td>3.00</td>
</tr>
<tr>
<td>0.55</td>
<td>3.180</td>
<td>3.18</td>
</tr>
<tr>
<td>0.60</td>
<td>3.331</td>
<td>3.33</td>
</tr>
<tr>
<td>0.65</td>
<td>3.466</td>
<td>3.46</td>
</tr>
<tr>
<td>0.70</td>
<td>3.521</td>
<td>3.52</td>
</tr>
<tr>
<td>0.75</td>
<td>3.673</td>
<td>3.68</td>
</tr>
<tr>
<td>0.80</td>
<td>3.757</td>
<td>3.76</td>
</tr>
<tr>
<td>0.85</td>
<td>3.827</td>
<td>3.83</td>
</tr>
<tr>
<td>0.90</td>
<td>3.910</td>
<td>3.89</td>
</tr>
<tr>
<td>0.95</td>
<td>3.947</td>
<td>3.95</td>
</tr>
<tr>
<td>1.00</td>
<td>4.000</td>
<td>4.00</td>
</tr>
</tbody>
</table>

Column Numbers

IV. CONCLUSION

In this paper we have presented an improved design of an M-to-B arbiter. We have also presented an improved design for a 2-to-1 arbiter. A faster arbitration circuit can be built for a multiprocessor multiprocessor system, using our M-to-B arbiters. In our design, for a fixed number of buses, the arbitration delay increases at a rate \( O(\log M) \). The logic circuits of our design are also very simple. Our arbitration circuit would be very attractive for a multiprocessor system with more than 4 memory modules, because both the cost and the delay would be less in our circuit as compared to those for other previously published designs. Our design could be easily fabricated to build an integrated circuit for a basic arbiter module, say an 8-to-4 module, a 16-to-8 module etc. These basic arbiter modules can then be used to a tree like structure to build an arbitration circuit for a system with large number of memory modules.

REFERENCES