Codes for a Priority Queue on a Parallel Data Bus

D. E. Wallis and H. Taylor
Communications Systems Research Section

The article describes some codes for arbitration of priorities among subsystem computers or peripheral device controllers connected to a parallel data bus. At arbitration time, several subsystems present wire-OR, parallel code words to the bus, and the central computer can not only identify the subsystem of highest priority, but can also determine which of two or more transmission services the subsystem requires. The article contains a mathematical discussion of the optimality of the codes with regard to the number of subsystems that may participate in the scheme for a given number of wires, and also the number of services that each subsystem may request.

I. Introduction

This article describes some codes for arbitration of priorities among subsystem computers or peripheral device controllers connected to a parallel data bus. At arbitration time, several subsystems present wire-OR, parallel code words to the bus, and the central computer can not only identify the subsystem of highest priority, but can also determine which of two or more transmission services the subsystem requires. This article contains a mathematical discussion of the optimality of the codes with regard to the number of subsystems that may participate in the scheme for a given number of wires, and also the number of services that each subsystem may request.

II. Mathematical Discussion

Consider m users strung out along a bundle of n wires, along which they send one of three demands each, to a central terminal. User i can demand action A_i, action B_i, or no action. Each individual wire carries one bit of information (0 or 1) to the central terminal, namely, *no user's signal* is on that wire or *some user's signals* are on that wire.

By way of coding we can design a black box for user i, with buttons A_i and B_i causing two preselected choices of signals for the bundle of wires. Pushing neither button will contribute the Boolean zero; pushing A_i will send one Boolean word of n bits (not all 0); pushing B_i will send another; pushing both A_i and B_i at once will be prevented by a mechanical contrivance inside the black box.

When the system is operating the central terminal is supposed to be able to "understand" every possible message it gets on the bundle of wires. Whatever Boolean word it gets it must identify the user of top priority in that word, as well as the demand of that user.

The example in Fig. 1 has users 1, 2, 3 on a bundle of four wires. Each wire is represented by a column in the figure. For each "button" there is a row representing the Boolean word of four bits that button will contribute. Thus, if user 3 pushes button B_3, user 2 pushes button A_2, and user 1 pushes button B_1, then the word at central will be $w_1w_2w_3w_4 = 0111$, and central will know that the top priority user 3 is "on" and specifically demanding B_3.

To show that the system always works we need an algorithm to analyze any Boolean word $w_1w_2w_3w_4$ which

might appear at the central terminal. One such algorithm is pictured by the decision tree in Fig. 2.

Now more generally we can try for the most efficient priority queue (PQ) on n wires. In the example of Fig. 1 we could handle one more user of higher priority than the others, allowing him just one button. The scheme in Fig. 3 shows a way of putting n users on n wires, giving two buttons each to all the lower priority users and one button to the top priority user.

A clearly defined combinatorial problem arises from the example scheme of Fig. 3. Under the given conditions, could any of the users be given more buttons? Arthur Rubin has given a proof that the answer is "no." Thus the optimality of the scheme in Fig. 3 has been proven.

Here is proof (due to Lloyd Welch) that if there are n users, then there must be at least n wires, and the top priority user can be given only one button.

We assume, of course, that each user has at least one button, and say that the users are $1, 2, \cdots, n$ with n having top priority. Consider the following sequence of possibilities.

(1) User 1 (lowest priority) pushes one of his buttons. For central to know it there must be at least one wire – let's call it p_1.

(2) User 1 is on p_1 and user 2 pushes one of user 2's buttons. There must be another wire p_2 just to tell that (higher priority) user 2 is demanding something.

(n) User 1 is on p_1, user 2 is on $p_2, \cdots$, user n-1 is on p_{n-1}, and user n pushes one of his buttons. As in each previous case, there must be another wire p_n different from the wires $p_1, \cdots, p_{n-1}$ just to tell central that user n is demanding something.

Finally, the top priority user cannot be given a second button because that would require yet another wire different from $p_1, \cdots, p_{n-1}$ and different from p_n.

III. Application

The PQ has been used as an arbitration code for parallel arbitration of data-transmission priority among multiple users (subsystem computers) of an optimized, 24-line, bi-directional, digital data I/O and control bus connected to a central computer.

On this bus, it was desired to time-share the data transmissions and the priority arbitrations on the same wires. The purpose of the arbitration by the central computer was to identify, from among many subsystems having simultaneous pending requests for data transmission, that particular subsystem whose priority was the highest at that instant. Further, the scenario for use of the data bus identified two actions (or services) that each subsystem could request:

(1) Data input (subsystem has read and unloaded its data-input register, and is ready for next input).

(2) Data output (subsystem has written and loaded its data-output register, and is ready to output).

It was then desired to *vector* these requests to the central computer, so that the central computer would not have to make any further tests to determine the direction (input or output) of the desired data transmission. It was evident that economy of wire usage would require the subsystem computer to determine, for itself, which direction of transmission was the more important at a given instant. Then, the subsystem would make an interrupt request, and would drive the bus at arbitration time with a parallel code word, the receipt of which at the computer would be sufficient both to identify the subsystem and to identify the desired direction of transmission.

In the bus design, it was desired to arbitrate priorities in a manner which was independent of the relative positions of the various subsystem computers, and the central computer itself, along the bus, i.e., independent of the *electrical closeness* of the subsystems to the central computer. Thus, the parallel, wire-OR connection for driving the arbitration code words onto the bus was adopted. It then became evident that it would be desirable to maximize the number of users that could arbitrate simultaneously on a fixed number of available wires. Thus, the combinatorial study of the various possibilities and their degree of optimality was undertaken.

IV. Conclusion

The number of wires in the data bus is becoming a major cost factor in computer and signalling systems, and it is highly desirable to maximize wire utilization. When parallel arbitration of transmission priorities is used, as for example on the Mod Comp[1] computers of the DSN, more arbitration information can be carried on the wires than is presently transmitted. With 16 data lines and one "request" line, the Mod Comp presently arbitrates 17 priorities, with (in the terminology of this article) one demand per priority. The scheme discussed here, when applied to the same case (where one wire of the 17 is a "request"), also permits 17 priorities, while also affording *two* demands to 15 of these priorities.

[1] "Mod Comp" is a registered service mark of Modular Computer Systems, Inc., Ft. Lauderdale, Florida.

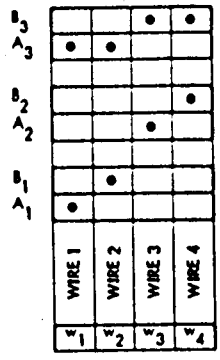

Fig. 1. Example of priority code on four wires with three users and two actions per user

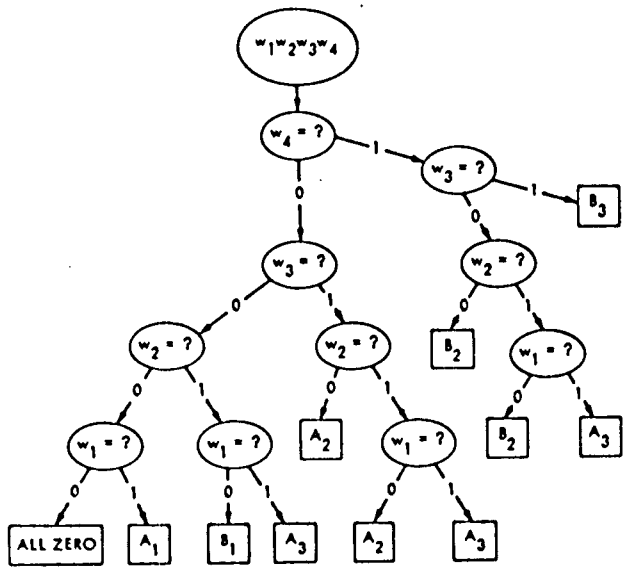

Fig. 2. Decision tree for decoding the example code given in Fig. 1.

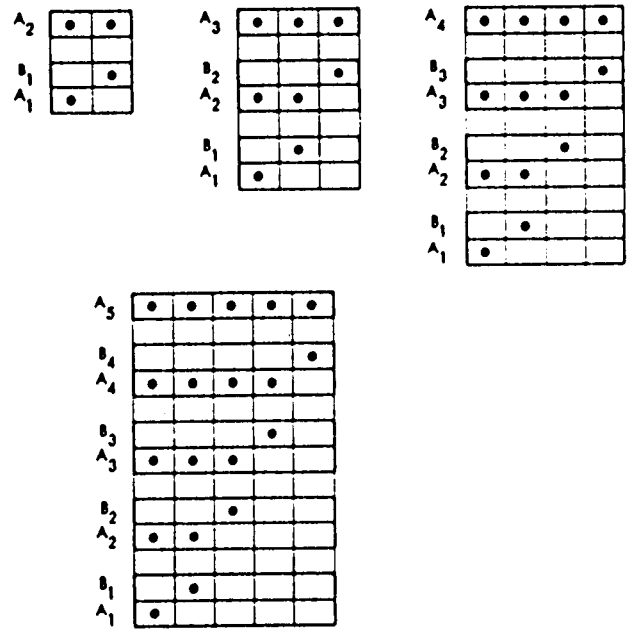

Fig. 3. Efficient priority codes that maximize the number of users on n wires, n = 2, 3, 4, 5