Comments on GBT Memo No 89:
"GBT Monitor and Control Design Rationale"
by M. Clark and R. Fisher, September 16, 1992.

Larry R. D'Addario
November 13, 1992

I have been fairly critical of some of the design decisions taken in the GBT monitor and control system and have been asked to explain these criticisms in writing. The present note is intended to accomplish that. It is assumed that the reader has a copy of the subject memo available for reference. It would be helpful, but not essential, also to be familiar with GBT Memo No. 88.

First, I do not intend to criticize the methodology or object-oriented approach or most of the general principles employed. For the most part, these are reasonable. My criticisms have to do mainly with the resulting design decisions. It is a little difficult to identify these design decisions in the available writings since they are buried within abstract statements of principles.

One overall criticism is that the present design is very non-quantitative and seems to have been made in something of a vacuum. Not only in the referenced document, but also in the 45-page Memo No. 88, there is nowhere a consideration of the time scales of anything. How can a real-time control system be designed without any quantitative consideration of the timing requirements? It is assumed that every controlled device will have time-of-day information. But to what accuracy? For each device, how long does it take to change state in response to a command? What data rates are needed to keep devices properly updated? It seems to me that a real design cannot begin until these numbers are known, at least approximately. The proposed design is far too abstract. I understand that this is intentional, in the hope that logical relationships can be established independent of any real hardware or quantitative considerations; but this is quite unreasonable, and has been carried to such an extreme that hardly any connection to reality remains.

The subject memo is just 2.5 pages long, and the first page discusses only general principles on which there is no disagreement. The second page lists six "assumptions," and these require some discussion.

"First, it is feasible and desirable to distribute the control software..." Some degree of distributed processing is certainly feasible, but what degree? The existence of modern networking software is cited, and the performance of this software and the corresponding hardware (e.g., ethernet controllers) determines *quantitatively* the extent to which real-time processing can be distributed. For example, it might be possible for a central controller to guarantee updated information to all peripheral processors once per second. It is probably not feasible to do this once per millisecond. But what is really required? What is desirable? This depends on quantitative specifications such as the data rates needed to update each hardware device, and these specifications are completely absent. The authors promise to "address the desirability of distributed processing..." but, by the end of the document, they never do. They seem to have assumed that distributed processing and networking are to be used for the same reason that

mountains are to be climbed: because they are there.

Now, I don't mean to imply that distributed processing is bad. Indeed, some very complex functions of the GBT might each be assigned to a separate computer for reasons of CPU loading, ease of development and testing, and for physical separation (minimizing wiring). For example, it seems fairly clear that the active surface needs its own control computer (as well as various subordinate processors embedded in range finders and other devices, although these should be regarded as part of the hardware). What's wrong is to assume, in advance and without any quantitative specifications on the requirements of individual devices, that distributed processing should be used as much as possible. Instead, each case should be considered individually and a separate processor should be allocated if and only if it is justified by the circumstances.

"Second, time-critical functions...must be isolated." The principles stated here are reasonable, but what is "time-critical"? Is an event that must be scheduled with a precision of 1 msec "time-critical"? How about 1 sec? How about 1 min? This makes a big difference in the design. But the statement, "The only synchronization mechanisms...in the M&C design are absolute time and common high-level command software" is not a *principle* or *assumption* but rather a significant design decision. This is not the only approach that adheres to the principle of isolation, and it should not be pulled out of the air and adopted without consideration of alternatives.

"Third, most of the...software needs only to be fast enough to provide an...acceptable delay..." Well, this is a trivial and obvious statement, but the sentences that follow do not logically flow from it. What is an "acceptable delay"? Once again, a quantitative analysis of the timing is needed before any design decisions can be made. Yet, in the vacuum of no such analysis, the profound design decision seems to have been made that UNIX and its standard networking tools can be used, in spite of having no guarantee of response times. It is well known that programmers at the NRAO are very familiar with UNIX and like it; could it be that this design decision came about from such bias? The argument might be: real time programming is difficult; UNIX is not a real time OS; the Sun workstation on my desk runs UNIX; therefore, let's design the control system so that there are no real time requirements on most modules; whenever a real time requirement arises, we'll force it into a separate computer. This reasoning leaves out *all* considerations of system requirements. It could easily have the effect of requiring all real-time work to be handled by the engineers responsible for the associated hardware devices, so that the real-time processors and their software are made part of the Electronics Division's responsibility and budget, rather than the Computer Division's. While the total system becomes more complex than necessary, this fact is hidden by the separation of responsibility. The formal M&C system becomes simpler only because the hard part has been thrown over the fence into the neighbor's yard.

I have been told (M. Clark, private communication) that such a shift of responsibility is not intended, and that the Computer Division will take responsibility for all real-time code. Nevertheless, unless reasonable, quantitative timing specifications are set at the beginning, much work will be forced onto the design engineer; given a special requirement in the hardware, the engineer is likely to handle it with a simple imbedded processor over which he has complete control, rather than trying to explain his needs to the Computer Division and have them fit it into their very complex

processor. (I have seen this happen in many other projects.) The result is that the complex "real time" computer (in which, remember, it is intended to have as little real time code as possible) has almost no real work to do.

The fourth ("software units...must be small...[with] a minimum of communication"), fifth ("new modules may be added...with minimal disturbance"), and sixth ("safety issues must be handled at the lowest software level") points are ones that I agree with. But under point four it is mentioned that distributed processing is compatible with isolation of software units; of course, this is true, and in fact it helps to enforce such isolation. But it is certainly not true that the isolation principle requires the use of distributed processing.

The penultimate paragraph of the document describes a major design decision which I believe deserves very strong criticism. The description, however, uses language that fails to convey the actual design decision. The authors call their design "asynchronous" or "delegated-authority," and they choose to label all alternatives as "micro-managed," perhaps imposing a bias based on the way the same terms are used for human management. What they have actually decided is that every device in the system must be able to accept commands that are to be executed at some specified time in the future, rather than immediately. In this way, the main control system is relieved of all real-time requirements. It can be learned from studying Memo No. 88 that this idea arises mainly as a way of dealing with the antenna motion, which is rather complicated and for which the completion of a particular command (arrival at a desired position) will often be many minutes in the future. Whether this is a good design for the antenna motion control is arguable (but I won't argue it here), but it is certainly not a reasonable design for the vast majority of other devices. It implies that every device, no matter how simple, must have a rather sophisticated computer capable of supporting the control system's communication protocol (presumably TCP/IP over ethernet, and therefore requiring a rather powerful processor and operating system) as well as its higher level command protocol and must also know the absolute time (which Memo No. 88 suggests should be distributed by the complex and expensive IRIG-B). In fact, the control of nearly all devices (front ends, local oscillators, "routers," and most back ends) consists of a few bits of switch settings and/or a few simple numerical parameters which can be effected within a few milliseconds (in some cases microseconds) of the receipt of a command. In nearly all cases this execution delay is completely negligible and can be ignored, so that the software could consider commands as being executed immediately.

If the software design allowed it, most hardware devices could have a very simple control interface. This would facilitate the adding of new devices in the future, especially those (like front ends) that require very little in the way of computer control. The proposed design forces all devices to the level of complexity of the most complex device (presumably the antenna motion). This is not reasonable.