Identification

Configuration Checker
J. H. Saltzer

Purpose

The configuration checker is a subroutine which is called to check (so far as possible) that the configuration information in the Major Module Configuration Table (Section BK.4.04) agrees with the actual hardware switch settings.

Specification

Calling sequence:

```
declare status fixed;
call check-configuration (status);
```

if all tests are successful, meaning that the contents of MMCT agree, as far as can be tested, with the actual hardware switch settings, then the argument "status" is set to zero. If any test fails, status is set to an appropriate non-zero value as given below. Check-config does not attempt to communicate messages about problems encountered except by its single status return.

In general, it is not possible for a program to directly sense the settings of configuration switches. Instead, it is necessary to perform indirect tests to attempt to verify the hypothesis that the Major Module Configuration Table is correct. There are, in fact, certain potential differences between the MMCT description and the actual configuration settings which are not detectable by program. These differences in general are not of consequence to the running Multics except in messages to an operator which could then refer to the wrong piece of physical hardware.

Check-config does not attempt to verify that major modules indicated as "not present" are indeed "not present". It only checks for correct configuration of those major modules claimed to be present in the system. Check-config operates in the limited EPL environment provided by Bootstrap I and II. (BC.4) That is, it may be written in EPL, it is called on a standard stack, etc. In making its tests, Check-config will save and restore any permanently assigned storage areas (e.g., GIOC mailboxes or CPU fault vector locations) which it must change. Certain subroutines of check-config must be written in EPLBSA.

Discussion

Following is a list of the specific tests made by check-config. With each test is given the status return if the test fails.

1. Test each connected system controller for proper addressing range.
2. Test each connected system clock for:
   - proper address
   - reasonable setting
   - counting
   - correct interrupt cell assignment
3. Test each connected GIOC for
   - proper base address
   - response
   - correct interrupt cell assignment
4. Test each connected Drum controller for
   - proper base address
   - read 1 record
   - correct interrupt cell assignment
5. Test each connected CPU for
   - proper base address
   - correct controller assignment
   - correct CPU tag.