Machine Protection System Algorithm Compiler and Simulator *

Gregory R. White **Gregory Sherwin**

Stanford Linear Accelerator Center, Stanford University, Stanford CA 94305

Abstract

The Machine Protection System (MPS) component of the SLC's beam selection system, in which integrated current is continuously monitored and limited to safe levels through careful selection and feedback of the beam repetition rate, is described elsewhere in these proceedings.

The novel decision making mechanism by which that system can evaluate "safe levels", and choose an appropriate repetition rate in real-time, is described here. The algorithm that this mechanism uses to make its decision is written in text files and expressed in states of the accelerator and its devices, one file per accelerator region. Before being used, a file is "compiled" to a binary format which can be easily processed as a forward-chaining decision tree. It is processed by distributed microcomputers local to the accelerator regions. A parent algorithm evaluates all results, and reports directly to the beam control microprocessor.

Operators can test new algorithms, or changes they make to them, with an online graphical MPS simulator.

Problem Statement and Rationale

The Machine Protection System (MPS) monitors, in real-time, potential autogenic operational hazards of the Stanford Linear Collider (SLC). These include the ambient radiation of the beam-pipe and its devices, vacuum, water and other critical parameters throughout the accelerator.

The MPS's purpose is to limit the integrated current of the beam to any part of the accelerator to safe levels, while continuing to deliver beam with the desired parameters to the rest of the machine; that is, to be minimally invasive. The extent to which the current is lowered should be just enough to make the SLC's operation secure, but not so low as to make the cause of the problem untraceable. If, for instance, a collimator is causing a hazardous radiation shower in one section of the SLC, MPS should tell the software system that selects beam configurations to lower the current in that sector. MPS must return the beam to its desired current and configuration automatically, as soon as it detects that the fault has been ameliorated.

Overall Solution Method

Other papers in these proceedings describe the MPS system in general [1] and two of its subsystems[2][3]. Here we summarize the overall solution strategy, so as to put the MPS Algorithm Compiler and Simulator in context.

Devices and parameters deemed critical to the operation of the SLC are connected via MIL-1553 to a VME based microprocessor local to an accelerator region. These microprocessors compute an MPS "algorithm". The algorithm processors or "AP"s are arranged in a two tier hierarchy, there being one Supervisor AP whose inputs are the outputs from the other APs and whose output is fed directly to the beam control computer, which then adjusts the next beam's parameters. This process is repeated with each beam pulse. See figure 1.

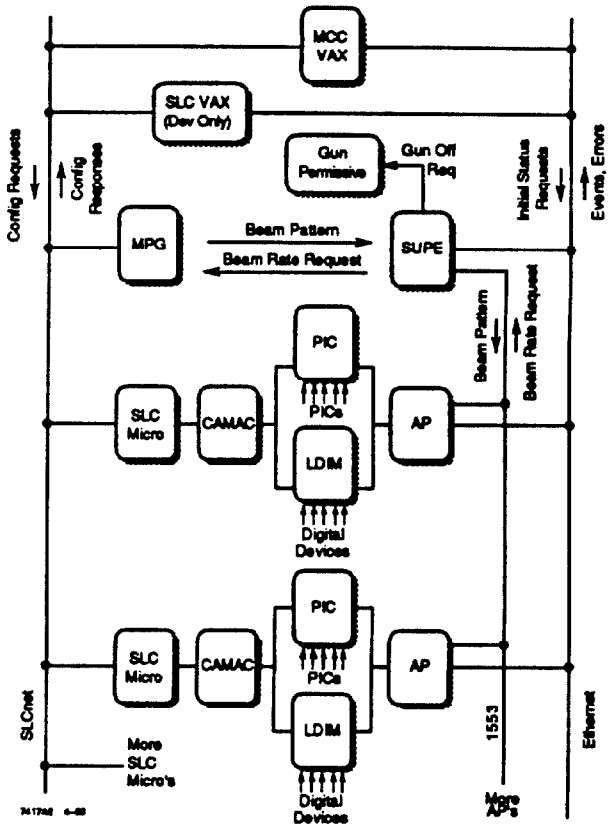

Figure 1: MPS Data Interconnection Diagram

Since the devices which form the input to an algorithm, and the available re-configurations of the accelerator which form the output, are different for each sector of the SLC, a different algorithm must be developed and tested for each algorithm processor. For that reason, the algorithm "compiler" and "simulator" software were developed. They are two distinct programs, their relationship being analogous to the compilers and debuggers used in conventional programming.

The following describes first the algorithm compiler (MPSL) and then the simulator (MAS).

*Work supported by the Department of Energy, contract DE-AC03-76SF00515

0-7803-1203-1/93$03.00 © 1993 IEEE

ALGORITHM COMPILER

First a computer language was developed in which the following could be formally expressed: *i.* the states of devices *ii.* the state of the accelerator, that is, where there is currently intended to be beam, and *iii.* possible alternative accelerator configurations, principally with regard to repetition rate. A device "state" is a device name in equality or inequality with one of the values that device can report. A well formed algorithm then describes, for all expected states of the accelerator in a region, the conditions under which some specified alternative machine configurations should be adopted.

```
AP = AP92
ALG = "Demonstration for PAC 93"

/*
** Algorithm contains one beampath, consisting of two bgrp names.
*/
BEAM = "SLCSLD" | "ESA"

    RATE = "FULLRATE" |          /* Full rate everywhere                  */
           "LIMHI_FN" |          /* E- beam in arc/ff limited to 10 Hz */
           "LIMHI_FS" |          /* E+ beam in arc/ff limited to 10 Hz */
           "LIMHI_FF"            /* Beams in both arcs limited to 10 Hz*/

        STOPCNF = MPST.AP92.1."PROT" == "UNPRTCTD"  &
                  MPST.AP92.2."PROT" == "UNPRTCTD"
                      ["A Stopper's OUT or MOVING"]

            NEWRATE = "ZERORATE" /* Absolute minimum needed to run SLC */

                VACV.LI31.114."VALVE" == "CLOSED" &
                PICS.LI31.102."CABLE" == "OK"
                ;

            NEWRATE = "LIMIT_HI"

                PICS.LI31.102."THRESH_D" == "OK" |
                PICS.LI31.111."THRESH_A" == "OK" &
                    PICS.LI31.112."THRESH_A" == "OK"
                ;

            NEWRATE = "FULLRATE"
                ; /* default */

        STOPCNF = MPST.AP92.1."PROT" == "PROTECTD" ["STOPPER IN"]

            NEWRATE = "ZERORATE"
                ; /* default */

    RATE = "LIMIT_HI"              /* Beam everywhere limited to 10 Hz. */

        STOPCNF = MPST.AP92.1."PROT" == "UNPRTCTD"

            NEWRATE = "ZERORATE"

                VACV.LI31.114."VALVE"    == "CLOSED" &
                PICS.LI31.102."CABLE"    == "OK"
                ;

            NEWRATE = "LIMIT_LO"

                PICS.LI31.102."THRESH_D" == "OK"
                ;

            . etc etc.
```

Figure 2: Extract from an MPS algorithm for an AP

AN MPS ALGORITHM'S STRUCTURE

In the SLC timing system, a "beamcode" describes a beam in terms of the accelerator devices necessary to propagate that beam, and the times in relation to a fiducial that those devices must fire. "Beamcode modifiers" define the repetition rate of a beamcode, and in great part, where in the accelerator complex beam from that beamcode can go. A Regional Beamcode "group" is a set of beamcodes for running the accelerator for a particular experiment. Call a disjunction of these groups then, a "beampath".

To identify exactly where beam in a given pulse will wind-up, one needs to add a statement of the states of the beam stoppers. Call a conjunction of stopper states a "stopper configuration".

Then, for some disjunction of beampaths, for some disjunction of stopper configurations and for some target repetition rate, a single Boolean expression in the states of local devices is sufficient to specify whether that target rate can be adopted.

An algorithm is a list of expressions in local devices, one for each repetition rate possible in each of the beampath/stopper configuration possibilities. Once loaded with an algorithm, an Algorithm Processor, within one beam-pulse (8.333 msec), determines which beampath and stopper configuration are in effect on that pulse, and starts to evaluate the expressions specifying the safety of each available repetition rate. It does this from first to last, until it finds an expression which is false, at which point it concludes that the associated rate is the highest it can recommend. This means there is one last repetition rate per clause which is always true (has no expression) and specifies the highest possible repetition rate in that part of the accelerator.

ALGORITHM COMPILER FUNCTION

The primary function of the compiler is to translate each expression into a bit-mask on the MIL-1553 port data that each device sends to its AP. The compiler prepares the data on a per port basis. It provides a bit-mask for the location of interesting data, and a second mask for the values of that data were the associated expression to be true. The compiler acquires the information about how the devices are wired from the SLC database.

The output binary file is isomorphically very similar to the input file. In addition it contains some information to help the AP configure its data acquisition process optimally.

ALGORITHM SIMULATOR

The function of the MPS algorithm simulator (MAS) is to verify that recently developed MPS algorithms will perform as intended. The simulator tests the software integrity of the algorithms by providing the capability to simulate any possible state of the accelerator and its associated hardware devices. To accomplish this, the software attempts to simulate virtual devices, and their trip conditions, to verify that the algorithm performs according to design. Other beam and accelerator characteristics are also simulated as inputs.

The great number of variables which the simulator controls requires an extensive configuration procedure. This procedure includes the selection of APs, algorithms, states for various devices, beam groups, rate-limiting kinds, and states for stopper devices (ie. stopper configurations). In addition to this complexity, the software is event-driven (via individual button pushes and item selections from lists) and not procedural, and therefore much of the simulator software is exposed directly to the interface-level software. This creates software vulnerability issues.

To prevent improper or incomplete selections and to keep track of the current configuration state of the simulator, the program implements a state-machine using global variables. Once the simulator is properly configured, the user can execute a simulation and view the results and/or dump them to a file. At the core of the simulator processing is the same algorithm loading and evaluation software used in the algorithm processors to determine device trips and requested machine states. Configuration information is converted into the equivalent of MIL-1553-level raw port data and beam data, as input to the AP's evaluation software. All input conditions and processing results are presented in the simulator output.

The simulator is designed so that the user can then make minor modifications to the simulator configuration and repeat the simulation. File inputs and outputs are also provided for most

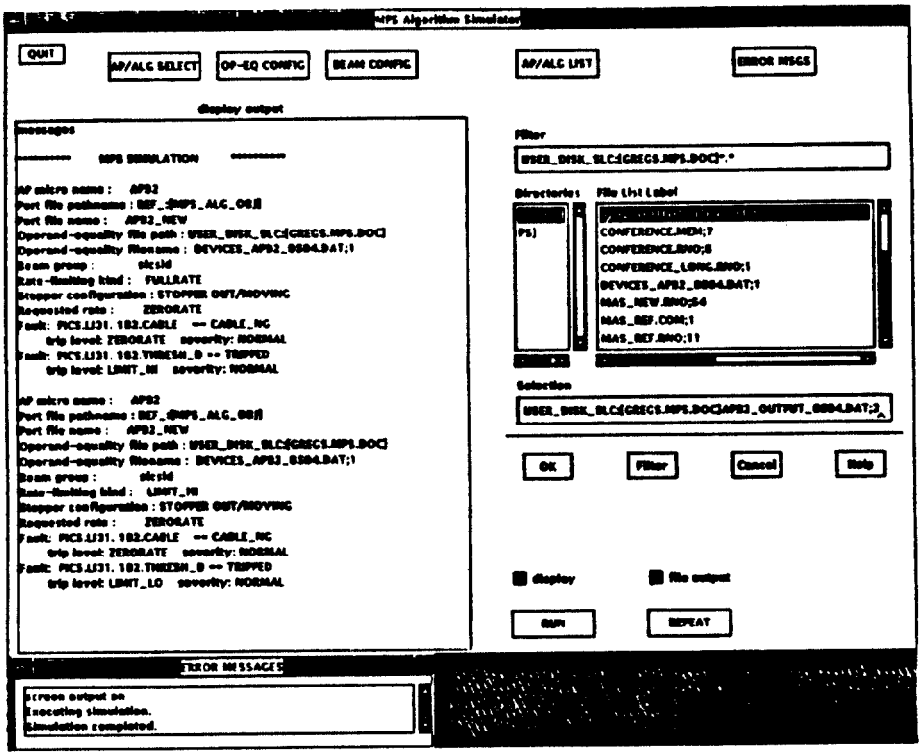

Figure 3: Algorithm simulator screen showing the result of a test when two devices were tripped

of the configuration data to simplify setup for re-simulations performed at a later time.

Since the MPS system is designed with devices subordinate to APs, which in turn are subordinate to a supervisor AP, there are several levels of operation which the simulator can perform. At the most basic level, devices are configured as inputs to a single AP. Additionally, individual APs can be configured as inputs to a simulation of the Supervisor and its algorithm. At the most complex level, devices can be configured as inputs to APs, which in turn are used as inputs to the Supervisor, hence simulating the entire MPS system from devices up to the Supervisor.

The simulator has a MOTIF user-interface consisting of various buttons, display windows, and selectable lists. To begin a simulation, the user must first select a mode of operation: the simulation of a single AP (which may or may or may not be a supervisor) and its direct inputs, or the simulation of the entire MPS system (a supervisor at the device-level). Once this selection is made, the user chooses an MPS algorithm and its associated AP from available choices presented in respective lists. From this selected algorithm file, a list of beam parameters and devices relevant to the AP is accessed and used to prepare their respective configuration portions of the simulator.

With an AP and an algorithm selected, the user can configure beam characteristics and virtual devices. Beam groups and rate-limiting kinds are activated for the simulation through selectable lists and button options. For devices, the user is presented with a list of devices and their currently configured states (ie. operand-equalities) to be used as input to the currently selected algorithm. To minimize the configuration effort, all devices are initialized to healthy states. Upon selection of any device, the user is presented with a list of possible states to configure the device to. To enhance the possible use of the simulator as a diagnostic tool, the user also has the option of configuring the list of devices to the states that exist on the running MPS system.

Once the simulator is configured, the user presses a button to execute the simulation and to acquire output. As illustrated in Figure 3 (cool screen capture), the output includes the algorithm processor name, the algorithm path and file names, the name of the saved file of operand-equality configuration data (if applicable), the beam group and rate-limiting kind, the stopper configuration, the requested rate resulting from algorithm processing, and individual device faults along with their trip levels and severities. Since a single simulation can be configured with multiple beam groups, rate-limiting kinds, APs and algorithms, the output is formatted by grouping the results for each combination of these inputs as if they were individual simulations.

The simulator has proved to be a useful tool in debugging the MPS software. Since the simulator used the same algorithm loading and evaluation software used in the real MPS system and was dependent upon the integrity of the MPS compiler, it served as a top-level software test platform from which to find and correct software errors in a variety of MPS software areas.

REFERENCES

[1] R. Chestnut *et al.* Machine Protection System for the SLC. *These proceedings.* 1993.

[2] S. Allison *et al.* MPS VAX Monitor and Control Software Architecture. *These proceedings.* 1993.

[3] K. Krauter, M. Crane. MPS Beam Control Software Architecture. *These proceedings.* 1993.