A Large (10,000 sensor) Fire and Gas Safety System implemented using polyFORTH and Execution Queues

R M Rodriguez and C L Stephens

COMPUTER SOLUTIONS LIMITED
Massillon Building No 2, Canada Road,
Byfleet, Surrey, England

A polyFORTH system has been designed that holds details of a large number of sensors in a database. Outstations using triplicated 6809s monitor these sensors and take shutdown actions if required reporting sensor status changes to a pair of VME 68000 based computers, one of which acts as a warm standby. The 68000 systems are responsible for processing these reports to produce operator displays, printed logs and to maintain fault reports. In the event of an incident the 68000 based system must absorb all of the reports being generated by the 6809s and react to them as appropriate. A major feature of any such system is invariably a queuing mechanism. This is used both to handle the input load and to prepare information for the 9600 baud terminals. In the process of developing such a system a mechanism called the execution queue was developed which was found to have many beneficial attributes. This paper will describe the application and the execution queue mechanism as well as highlighting the advantages of this powerful tool.

The equipment consisted of:

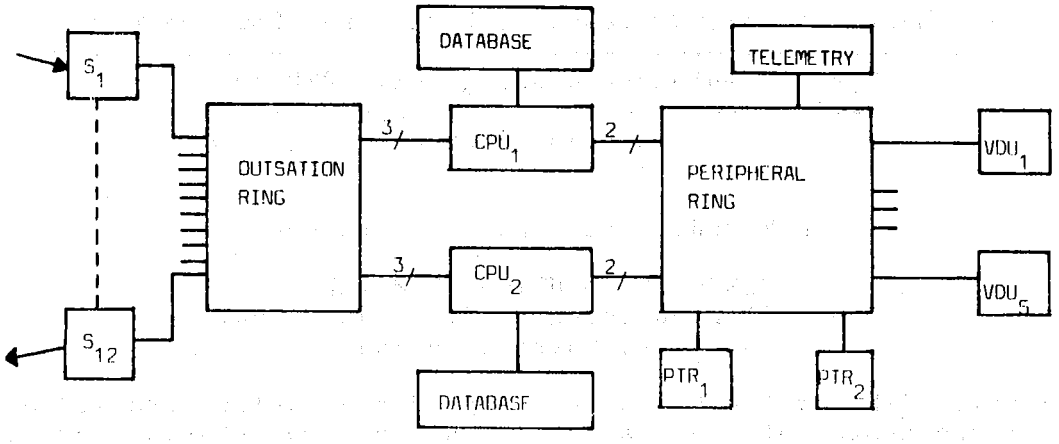
- Twelve remote plant outstations (6809 based) capable of interfacing with up to 10,000 I/O elements: sensors, valves, pumps, fans, etc.

- A high speed serial ring to report plant changes to a central processor. The ring is triplicated for improved security.

- An Active/Standby dual central processor sub- system to process plant events and operator requests. These are 68000 VME-based with 40 Mbytes Winchesters and 1 Mbyte of RAM.

- A high-speed serial ring to report outstation messages and general man-machine interface functions. This is duplicated for improved security.

- A number of peripheral devices for operator interaction:- 5 colour graphics VDUs, 2 printers and a telemetry link.



The system software is capable of:

- Plant status acquisition and storage.
- Event monitoring and alarm logging.
- Colour graphics status display under operator control or automatically on events.
- Operator interaction and configuration changes.
- Diagnostics and standby facilities.

Each one of these functions was assigned to a specific polyFORTH task and a task is defined to control each peripheral device.

The system includes a description of all plant I/O, events. alarms, and graphics representation. This is contained on a named-file, fixed-length record Data-base held on disk and in global RAM. Information in the data base is defined according to the particular function, and plays a key role in the performance of the software.

The operation of the system is illustrated by the processing of a plant event:

- When a plant I/O change takes place, a report is generated by the outstations., and sent to the central CPU. The report includes information on sensor identification (address in I/O space) and new value/status data.
- The report is received by the ring controller task which then commands the Exception-Report task to process the incoming data. It is then able to receive another report.

- The report processing task performs two functions: Update the data-base entries affected by the report and log the event in an appropriate alarm file.

  Any change to the data-base is reported to the operator (displays, printouts). To do this the report processing task in turn commands ALL operator interface task to update their output to show the change, and attract the operator's attention. Similar requests are sent to the Standby link task so that changes in the Active processor are also reflected in the data-base of the Standby CPU.

- The main interface with the operator is through the Display-controlling tasks. Plant status is represented as detail and overview mimic diagrams, histograms, Help-screens etc. The command to display the data-base change is vectored according to the "current" display type in each task. The first thing it normally does is to see if the sensor is currently being displayed, if not, no screen action is required. The characteristics of the display update (colour, attribute, symbol etc) are in turn vectored according to user-defined I/O element characteristics. This means that the event processing, storage and display update for all output peripherals is very fast (no IF .... ELSE .... THEN statements). If previously programmed and enabled into the I/O characteristics in the data-base a new display is automatically set-up in each task (auto-paint).

- In response to plant event reports, the operator can interact with the system in a number of ways: accept an alarm (by name, file position or time), isolate I/O elements from scanning by the outstation, re-direct task peripheral assigments, etc.

The system recognises three levels of access: Operator, Engineer and Supervisor, each level being a unique polyFORTH vocabulary with password access. This enables the use of the standard polyFORTH keyboard interpreter as the main man-machine interface. Operator options include display-type selection, Diagnostics, on-line editors (for data-base and displays) as well as on-line reconfiguration. All these options are protected against operator misuse with copies of operator actions being automatically logged.

When a fire occurs it is likely that a number of sensors will raise alarms simultaneously. The processing of these events requires multiple accesses to disk to interrogate the data-base as well as making entries in log files. The resulting display changes are output over the networks at 9600 baud. Both these actions will limit the short term throughput of the system. A common technique for simplifying the organisation of such systems is to separate the respective functions using a queue to act as a buffer

desynchronising the tasks. This tool also provides a solution to the problem of how to produce a multi window VDU using a conventional single channel terminal. In this case each data source (e.g. time display, sensor colour control, alpha log) places a suitable entry into the queue and a single task removes entries, examines the entry and performs the appropriate output.

Early on in the design of this system it was clear that many different types of message would be passed between tasks using the same queue. The scheme implemented consisted of words that would place a number of stack entries into a queue followed by the number of entries and the execution vector required by the destination task to process that data. For example:

```
: .LOG ( r F - )  2 :Q  LOGS ;
```

will place two values from the stack (record number, file address) and the address of LOGS onto the current queue. Typically LOGS will be defined as

```
: LOGS  ( r F - ) F# !  READ  .RECORD ;
```

where READ read the record number of the file whose address is in F£ and .RECORD prints the currently accessed record.

The destination task is now simply

```
BEGIN  QLENGTH  @
       IF  @QEXECUTE  THEN  PAUSE
AGAIN
```

Where @QEXECUTE unloads the queue prior to executing the vector.

This technique has proved very effective. Firstly it means that all serial output tasks including the standby link task execute the same code shown above. Secondly, as soon as the basic task structure is built, all application and terminal specific software may be developed interactively at the operator's VDU using conventional stack techniques but using the keyboard interpreter to transmit the desired queue entries to the task under development. Thirdly, the absence of flags or indexes with tables of actions means that as changes in the system are made, either to develop or modify the code, a change in one task automatically transmits its requirements to the destination task. This minimises coupling between the tasks giving a higher reliability when making program changes.

The complete software for the system took 12 man-months to design, program, test and document. The software forms the basis of a general package currently being installed in several other major control systems.