Mission-Independent Telemetry Processing Software for PCs

Richard J Miller, Director, Micro SciTech Ltd, Hollingworth-Court, Maidstone, England, ME14 5PP, Tel +44 622 663220.

Keywords
Telemetry Processing, Software, PC, RS232

Abstract
Until the early 80's, telemetry processing systems were commonly run on mainframe or mini computers running proprietary operating systems and software with limited portability. The advent of the 'low-cost' workstation reduced the hardware cost but the software still remained relatively expensive and relatively mission specific. The workstation itself, although comparatively cheap, was not, and is still not, an everyday piece of computing hardware. Telemetry Processing software has been developed by Micro SciTech to meet both low-cost hardware requirements and mission independence. It runs on networked IBM PC compatible computers and can be re-configured and used for many different missions and experiments without the need for extensive software rewrites.

Introduction
The real-time or near real-time monitoring of a mission payload, as regards end-users such as the experiment collaborators, generates the following key user requirements of a telemetry processing system:

- Frame Synchronization
- Data extraction and decommutation
- Engineering units calibration
- Alarm limit checking
- Parameter recording
- Plotting and display
- Data completeness
- Commanding
- Data archiving and retrieval
With so many missions and experiments being performed within a short time frame (<5 years) there is also a requirement for:

- mission independence

All these requirements can be implemented in software and, coupled with the need to minimise hardware costs, have been implemented as a serially networked telemetry processing system running on DOS based IBM PCs or compatibles.

By designing software with the goal of minimising processor load and, through the use of both standard and innovative techniques, the majority of software code can be written in the C high level programming language and satisfactorily implemented on computers as old as the Intel 8086 based IBM PCs. Furthermore, if the incoming telemetry data conforms to the RS232 standard, the data can be read and retransmitted using the serial port as supplied by default on most, if not all, PCs. The use of a second serial port enables many PCs to be daisy-chained forming an inexpensive networked system. The resulting system can process incoming data rates to 115Kbits/sec and serve one, ten or even one hundred users for what is essentially the cost of a few PCs.

System Description

Software Design Overview

With a strong emphasis on modularity both for maintainability and, perhaps more importantly, for future migration from a DOS based single-tasking operating system to a full real-time multitasking operating system, the overall system is split into the following seven tasks The data flow between tasks is shown in Fig 1.

- Executive
- Data I/O
- Packet Router
- Display Processor
- Configuration Processor
- Command Processor
- Maintenance

Whilst on a multitasking system each task could be executed as a separate child process of the executive parent, when executed

Figure 1 Task Data Flow
under the PC's DOS, they are actually called as subroutines from a single active DOS program forming the executive.

**Executive Task Control**

The executive controls the execution of the six other tasks and, since DOS is single tasking, is free to repetitively loop at the fastest rate the host PC will allow. However, this has been found to be unnecessary and therefore, with a view to multi-tasking migration, the loop rate can be slowed down such that the synchronous Packet Router and Display Processor are scheduled on a periodic 10 msec loop or sub-multiple thereof. This figure is dependent on many factors but, broadly speaking, it should be higher than the incoming telemetry frame rate. With a nominal period of 10ms the frame rates should be less than 100Hz, with 10Hz more typical. The 10ms value can be changed in real-time if it is found to be too fast or slow.

Apart from the synchronous Packet Router and Display Processor, all other tasks are asynchronous and executed as background tasks once the high priority synchronous tasks have completed. Since the executive actually cycles in much less than 10 ms, these asynchronous tasks are executed many times in the background before the executive re-schedules the synchronous tasks.

Although this task-scheduling system is, by nature, a true real-time multi-tasking problem, DOS is not! Furthermore, without resorting to BIOS programming, the PC timer resolution is limited to 18.2Hz which manifests itself in timer ticks of 50ms and 60ms. Thus it may appear that any figure below 50ms is unattainable.

To overcome the relatively poor timer resolution problem, an averaging method is employed. This averaging has an analogue in the 60Hz power supply generation whereby short-term fluctuations in frequency away from the 60Hz are compensated for by long-term high stability. Similarly, TPS loops with a long term stable average of 10ms but, on a microscopic scale, loops with a markedly fluctuating period that can be vary anywhere between 1 and 10ms. The loop-time itself can also be adjusted; on an 8086 XT, the system can only loop in around 100ms and so the 10ms period has to be increased to nearer this figure. An 'Autotune' feature can actually automatically adjust the loop-time for the user.

**Data I/O and the Interrupt Drivers**

All I/O is interrupt driven as polling for incoming data is inefficient and a waste of CPU resources. Whenever a byte of data is received or transmitted, a hardware
interrupt is generated and the Data I/O task reads or writes data to or from internal system buffers.

The I/o system has been flexibly developed with a C programming language interface to allow for the incorporation of custom-specific I/O routines to handle various network protocols. The default system accepts RS232 data streams but others, for instance, Ethernet TCP/IP network packets, can also be read.

By direct programming of the PC UART, processing of incoming data at rates up to 115 kbits/sec has been achieved. At this high rate, which approximately converts to 10 Kbytes/sec, an interrupt is generated every 100 micro-seconds. Using the lowest specification hardware platform (an 8086 PC) this approximately equates to the execution of only 30 instructions and thus requires some tightly written interrupt drivers to handle concurrent input and output. Full interrupt-driven serial communications software was written in 8086 assembler with higher level I/O management functions, required to rescue dying interrupts (a problem with interference and high data rates) and inform users of transmission loss, were moved to the background Maintenance task.

Packet Router

The Packet Router receives incoming data packets, performs some pre-processing, and then routes them to other users on the network and also to the display and, optionally, the local hard disk.

A packet consists of a header plus a variable length data packet. For the first release of the system, all packets only contain telemetry data and thus a 'packet' is a synonym for 'frame'. Future versions will, however, enable multiple packet formats so the system can process telemetry, command, voice and video packets all under one common system.

Packet pre-processing includes the standard data integrity checks such as checksum computation and packet sequence verification.

Packets can be input from previously archived disk files and need not be directly telemetred. Archiving is also performed by the system with data stored in one of three file formats: rawbinary, compressed or ASCII hexadecimal. The ability to replay mission data is useful for postflight analysis and operator training. The replay speed is also variable allowing, for instance, a complete month long mission to be replayed in a few minutes or stepped through one data packet at a time.
Display Processor

The Display Processor decommutates the packet parameters, optionally performs engineering units calibration, and then sends them to the display.

Telemetry displays tend to be textual, displaying message strings or numbers such as scientific parameters. For this reason, and to keep processor overhead low, the system works on a default text-mode display of 25 rows x 80 columns x 16 colours which is switchable to 640 x 480 pixel x 16 colours for real-time plots.

Every parameter within a telemetry frame is displayed in its own window termed a 'view port'. The system supports the simultaneous display of up to 256 parameters on a display page with up to 10,000 configurable pages per PC. A typical display page is shown in Fig 2.

![Figure 2 A typical display page](image)

With up to 256 parameters simultaneously being displayed and updated, the Display Processor reduces CPU overhead with three stages of optimisation. These stages are executed by the three hierarchical subfunctions of the display task, namely, the 'Change Checker', 'Display Scheduler' and the 'Draw Optimiser'.

**Change Checker:** This performs a simple comparison of the raw parameter's byte between its current and previous values. If a value hasn't changed since it was last displayed, it is not redrawn.
**Display Scheduler:** This function basically controls the parameter display refresh rate. With the system loop-time set at 10ms, the fastest rate at which a parameter can be displayed is also 10ms. However, most parameters do not change at this high rate. Furthermore, the eye cannot detect such rapid changes. It is therefore pointless to display a task at 100Hz and, using the Page Display configuration files, a lower rate can be set by the user. For the purposes of recording calibrated parameters, however, parameters may still need to be processed at 100Hz, even if they are not displayed. This is one reason why the display task itself is called at such a high rate.

If a parameter is not filtered out by the Change Checker and Display Scheduler, it finally passes though to the Draw Optimiser.

**Draw Optimiser:** This function inhibits the display of redundant text. After a parameter is scheduled for display, before being sent to the low-level display routines, it is converted to an ASCII text string representation (all low level display works on text strings only). The draw optimiser intercepts this string and actually transmits to the Basic Input/Output System (BIOS) only those characters that are different from the prior displayed text. However, there is a trade-off between the extra processor overhead incurred by this semi-intelligent Display Optimiser and the overhead of the BIOS in blindly printing all the characters in a brute-force method. For this reason, the Draw Optimiser is a display option which can be switched on and off by the operator.

An alarm system is incorporated within the Display Processor to monitor the calibrated parameter values. It enables a parameter, outside of limits, to be logged to disk with an accompanying sounding alarm and flashing display.

**Command Processor**

Commanding of the system, itself, for such purposes as changing a page, setting an alarm and recording or replaying data, is all done via Common User Access (CUA) style menus and dialog boxes whereby a command is triggered via an 'ALT-key' sequence as is common to several DOS applications. This usually brings down a further menu with more options selected by a single hotkey depression.

Actual Mission control through telecommanding is handled using a custom written command compiler incorporated into TPS via a C programming language interface (as for the telemetry interface when using I/O systems other than RS232). This allows the system to handle a multitude of different command systems.
Maintenance Processor

The Maintenance task executes repeatedly in the background, performing diagnostics and providing internal health-check data. All monitored parameters can be displayed as for telemetry parameters. The system attempts to off-load as much basic diagnostics as is possible by performing a complete pre-load checkout of the host machine and all accompanying software prior to loading. At its most mission-critical, the pre-load check will only permit TPS to load if it passes ALL checks. This has shown by experience to eliminate many run-time problems caused by software corruptions (intentional or otherwise) and hardware incompatibilities.

Mission Independence and the Configuration Processor

Mission independence is achieved by using flat-format ASCII text files to specify every aspect of the system configuration for a mission. The Configuration Processor task reads these files upon initial system-load or online page change and re-configures TPS appropriately.

Major changes between missions usually involve a new telemetry frame format and several new display pages. Since telemetry frame formats, display page and system setup are all specified in the text files, adaptation of the system for a new mission only entails editing these configuration files.

A simple language is used for attribute specification and takes the form, \textit{identifier} = \textit{option}, e.g., \textit{backgroundcolor} = \textit{red}. It is deliberately kept very simple and has a natural language, nested block-structured form so that, for instance, all value-port display attributes are specified within a 'valueport' block. The valueport block, itself, has separate sub-blocks for specification of attributes such as the calibration method to be used.

As regards reading incoming telemetry frames, only the packet header need be specified. The data packet, suffixed to the header, can remain completely unspecified other than its total byte-length which is embedded in the header and can dynamically change throughout the mission. Telemetry frame format is reduced to merely specifying the synchronization pattern, frame counter and sumcheck information.

For displaying and recording parameters, their start byte position in the data packet, byte length (or bit length) and data-type must also be specified in the configuration files. Acceptable Data types form a superset of the C language in which the majority of the system is written. Binary and multiple byte hex dump formats are commonly
required display formats but are not featured in the standard ANSI C language and have thus been added as extensions.

Conclusions

A multi-user telemetry processing system has been implemented on a low-cost network of PCs and performs the majority of standard tasks as required for small to medium mission monitoring. Incorporation of a three tier display optimization process permits high refresh-rate displays (10-100Hz) of many different parameters. With custom designed serial data interrupt drivers, fast incoming data streams to 115 Kbits/sec, adequate for a large variety of experiments, can be read and re-routed without data loss. Specification of the complete mission configuration using only text files removes hard-coded mission specificity thereby eliminating software rewrites and reducing mission configuration to a simple editing task.

Acronyms used

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>ANSI</td>
<td>American National Standards Institute</td>
</tr>
<tr>
<td>BIOS</td>
<td>Basic Input/Output System</td>
</tr>
<tr>
<td>CUA</td>
<td>Common User Access</td>
</tr>
<tr>
<td>DOS</td>
<td>Disk Operating System</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver and Transmitter</td>
</tr>
</tbody>
</table>