SAND2000-2021C

Taming the Viper:
Software Upgrade for VFAUser and Viper

Randall Takeshi Dorin
University of Arizona, Tucson, Arizona

Technical Advisor: John Moser III
Manager: Michael E. Partridge
Telemetry and Instrumentation: Dept. 2665
August 8, 2000

Abstract

Due to memory card malfunction, the Viper has previously been unable to record meaningful data. In order to improve the Viper's reliability, it has become imperative to create a function to test for faulty memory cards. The Memory Test function will indicate faulty cards by number after the test has been completed successfully. This function will also be integrated with existing code in Delphi, C, and Assembly languages, manifesting itself as a button on VFAUser.

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Contents

Improvement Summary

This report describes the procedure and properties of the software upgrade for the Vibration Performance Recorder. The upgrade will check the 20 memory cards for proper read/write operation.

- The upgrade was successfully installed and uploaded into the Viper and the field laptop.
- The memory checking routine must run overnight to complete the test, although the laptop need only be connected to the Viper unit until the downloading routine is finished.
- The routine has limited ability to recognize incomplete or corrupt header and footer files.
- The routine requires 400 Megabytes of free hard disk space.
- There is one minor technical flaw detailed in the conclusion.

Taming the Viper:
Software Upgrade for VFAUser and Viper

Memory Problems with VFAUser

Currently, a large problem with the Viper system is that on occasion, one or more of the Flash memory cards fail. The Viper currently saves 16 bits of information sequentially across 20 Flash memory cards. If one or more of the cards fails to record the information correctly then the test will result in useless data. Unfortunately, neither VFAUser nor the Viper has any way to check the operational ability of the cards.

Background Information on Viper and VFAUser

The Viper system is a vibration sensor attached to the back of bombs to determine the amount and direction of force felt during aerial maneuvers. There are two separate parts to the Viper system: the Viper unit that is attached to the bomb, and a Microsoft Windows interface named VFAUser used to communicate with the Viper unit. VFAUser communicates with the Viper unit through a field laptop's parallel and serial ports. Using these ports limits the speed of communication, but that cannot be easily changed.

The Viper Unit itself has two separate parts contained in a metal shell. One part, known as the Memory module, holds twenty 20Mb Memory cards and acts primarily as a place to store data. The other part, known as the Control Unit, consists of a Motorola microprocessor and RAM. It' as it's name implies controls and regulates the Viper Unit.

The Viper Unit was originally programmed in Assembly, and then later translated and improved into C for clarity by John Moser III. VFAUser was written by Daniel Gallegos in Delphi 5, an Object Pascal environment. It is

necessary two write in all three languages to add any new function to the Viper system.

Upgrade Goals

The goal of this Viper Upgrade is to create a function that will test the Flash memory cards' ability to correctly record and download data from the sensors. It should have the following properties:

- Be able to identify the bad card by name so that they may be replaced
- Check the entire memory of all 20 cards.
- Appear as a button on VFAUser under the "Memory Module" tab.
- Finish Operation in a timely manner.

Methods and Materials

Matching expectations

The first step in the process of engineering the software upgrade was to verify that both the programmer and the consumer had matching expectations as to what the final specifications of the software upgrade would be. In order to ensure and document the agreement, the programmer created two documents: a Statement of Work and a Requirements Document.

The scope of the two documents is slightly different. The Statement of Work encompasses a broader-spectrum than the Requirements Document in that it includes a general description of the requirements as well as a budget and milestone summary. The Statement of Work therefore establishes the scope of the project in both time to create the upgrade and expectations of the final product. This is useful in that it creates a scale by which to measure progress.

The Requirements Document focuses on what the upgrade will do, and how the upgrade will be activated. In short, the Statement of Work establishes the

scope, and the Requirements Document fills in the detail creating the substantive description of the desired abilities of the upgrade.

Combined, the two documents outline what is to be delivered, and how long it will take to deliver it. Once created, these documents are sent to the consumer to ensure that they are compliant with consumer expectations.

During the creation of the upgrade, the programmer was in close contact with his supervisor, meeting at least once a week to discuss progress and problems. During these meetings, expectations for the software continually evolved due to previously unseen problems as well as new insights.

Learning New Languages

The requirements for programming the new code include an understanding of three different programming languages: C, Object Pascal and Assembly for the Motorola Micro-controller. Unfortunately, the programmer only knew C, and Pascal. In order to program the upgrade it was necessary for the programmer to learn Assembly and Object Pascal (as it pertains to Borland's Delphi 5) well enough to accomplish the upgrade goals.

This upgrade involved adding functionality instead of creating a better version of the software, therefore there were two ways to learn the language: through reading, software tutorials, and instruction; and through reading existing code. As it turned out, the two most useful were instruction from John Moser and reading the existing code.

Writing New Code

Before the programmer can begin to create the new code, it is necessary to read and understand all the relevant pre-existing code. John Moser, creator of the old C and Assembly code for the Viper Unit, proved vital in pointing out the important aspects of the code. He often provided tips on how to alter the structure of the old code so that it would not interfere with the new code.

Although the code could be compiled on the machine that it was written, it could not be run. This is because in order to work properly, any new Delphi Code needs to be transferred onto a laptop that runs MS Windows95. Any new C and Assembly code needs to be uploaded into the Viper's memory. Therefore, most of the newly created code could not be tested unless the programmer went into the secure area. However, the programmer did not have the necessary security clearance hence any testing required the presence of his supervisor.

Due to the fact that the checking routine would be run solely on the laptop, the programmer decided that it would be best to write this part as a stand-alone program, therefore it could be run without going through the trouble of testing in the secure area.

Operational Overview of the Memory Test

When the memory checking routine is activated from VFAUser on the field laptop, the memory checking routine will send a message through the parallel port that will tell the Control Unit in the Viper to execute the associated command. The Control Unit will begin by erasing any data currently on the memory cards. Once it has finished it will send a message back through the parallel port that informs the user that it finished erasing the memory. The Control Unit will then immediately begin writing a known sequence of hexadecimal characters into memory. Once the Control Unit has finished, it will send a message back through the parallel port, and wait for further commands.

While VFAUser is waiting for the Control Unit to send a message indicating that it is finished erasing, it displays 'erasing memory' and a timer. Once it has received confirmation of a successful erasure, it displays 'writing to memory' and a timer, until it receives confirmation that the write has been successful. Once it has received confirmation of a successful write, it sends a command telling the Control Unit to begin downloading all the data through the parallel port into a predefined file on the laptop.

Once the Control Unit has finished downloading all the data into the laptop, it stops and waits for further commands. In contrast, VFAUser immediately launches into a checking routine that compares the known sequence of hexadecimal data to the downloaded data. If any irregularities are found, the checking routine calculates and displays what card the error came from, what byte it was, and what value was found. The checking routine then continues to finish checking the remaining data. If no errors are found, VFAUser erases the 400-Mb file. However, if errors are found, VFAUser saves the 400-Mb file and the file containing the error information displayed earlier.

Conclusions

The objective of the added function was achieved with one minor flaw: the checking function will report 32 corrupt bits if the first 16 bits of a header or footer is corrupt. This is minor, because the flaw accurately reports the error, but inaccurately reports the size of the error. Because any error must be manually checked to find the cause, it should be quickly noticed that there is only 16 corrupt bits.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.