D19-17

N87-15348

Software Development for the VLA-GDSCC Telemetry Array Project

H. W. Cooper
Radio Frequency and Microwave Subsystems Section

L. R. Hileman
Telos Corp.

Software for the VLA-GDSCC Telemetry Array (VGTA) Project is being developed in a new manner. Within the Radio Frequency and Microwave Subsystems Section, most microprocessor software has been developed using Intel hardware and software development systems. The VGTA software, however, is being developed using IBM PCs running consumer-oriented software. Utility software and procedures have been generated which allow the software developed on the IBM PCs to be transferred and run on a multibus 8086 computer.

I. Introduction

Intel was the only supplier of development hardware and software when the Radio Frequency and Microwave Subsystems Section first began using the Intel 8080 microprocessor for the control of electronic equipment. Over the years, several groups within the section have continued investing in and upgrading their Intel software development products.

While the Intel development system did serve its purpose, it was not a general purpose system in that both the hardware and the software were unique to Intel. New employees needed training on the system, and the system could not be used beyond the development of software because the Intel development system hardware could not be used as a target system.

II. Software Development for Parkes-CDSCC Telemetry Array (PCTA)

The PCTA Project (Ref. 1), forerunner to the VGTA Project, used integral microprocessor computers to perform the monitor and control function of all of its electronic assemblies. In addition, an "array controller" performed as a message switcher by interfacing two CRT terminals to the various assemblies. While the assemblies all ran ROM-based firmware, the array controller contained floppy disk drives which ran the CPM-86 operating system that enabled the array controller to read and execute command sequences from the floppy disks.

When the PCTA project was started, the implementing group did not have the required Intel development equipment.

As an alternative, the PCTA software, written in Pascal, was developed in the "array controller" computer which ran the CPM-86 operating system. This procedure totally eliminated the need for Intel development products. While this method worked, it had difficulties. The Pascal MT+86 compiler errors were numerous, and much time was spent finding work-arounds for them. In addition, claims for easy ROM programmability using Pascal MT+86 were unfounded, and assembly language and 8087 support was poor. The system was floppy-disk based, resulting in slow compilation times. One application program exceeded 64K in size, which required a complicated work-around in order to run a ROM-based system.

III. VLA-GDSCC Telemetry Array (VGTA) Method

What was needed was a hardware/software development method which was cost effective, well supported, free of errors, and efficient. After considering various options, the IBM PC system was selected since it used the same microprocessor, and because of the large availability of off the shelf software.

Given the development hardware, Microsoft "C," a high-level programming language, was selected for the following reasons:

(1) Microsoft wrote the operating system for the IBM PC and would be expected to be capable of writing a "C" compiler for it.

(2) "C" is a high-level structured language presently very popular in industry and in universities. The language includes a large standard library which performs most needed functions including port I/O, string functions, and memory management. The only problem initially noted with Microsoft "C" was its lack of real-time interrupt support, which has been solved by the development of JPL Software routines.

(3) Microsoft "C" supports code and data memory requirements larger than 64K, and also provides 8087 numeric data processor support.

(4) Microsoft also supplies a compatible macro assembler and symbolic debugger, both compatible with their "C" compiler.

(5) The Microsoft products are well supported and are available off the shelf.

Once the development method was chosen, the main problem was how to make software designed to run on an IBM PC run on an 8086 multibus system. Microsoft "C" didn't make any claim to ROM programmability of their code, and Microsoft "C" runs under the PC DOS operating system. The problems to be solved included how to transfer the code from the IBM PC to the 8086 computer, how to debug the software, and how to handle real-time interrupts. These problems have been solved through the use of utility software written at JPL. The utility software is used to transfer the compiled application program from the IBM PC environment to the target system. Transfer methods include floppy disks, PROMs, or downloading to RAM.

The "C" language (including Microsoft "C") itself does not provide interrupt handling. Even though all "C" functions are re-entrant, they end with a normal return, not an interrupt return. This problem has been solved by using simple assembly language routines which vector the interrupt, to the desired "C" routine, and then perform the required end-of-interrupt operations.

The VGTA retains the same monitor and control philosophy as did the PCTA. The ability to read and execute command sequences from the floppy disks was lost with the removal of the CPM-86 operating system from the array controllers. Therefore, it was decided to use the IBM PC as a smart terminal as well as a software development tool. Once again, selecting Microsoft "C" proved beneficial because it allows the IBM PC and Target Machine software to be written using the same language.

IV. Development Process and Required Equipment

Software development can be described by the following steps:

(1) Design the application program.

(2) Code the program.

(3) Compile the program.

(4) Test the program.

In the preceding steps, all but testing can be done on any IBM PC. Initial testing of programs, if no real-time input/output is performed, can also be done to some degree on the IBM PC. But in general, testing must be done on the target computer. It is for this reason that the utility programs were written.

The following hardware and software were required:

(1) IBM PC or equivalent with PC DOS.

(2) Microsoft "C" and Microsoft macro assembler.

(3) Target system consisting of:

(a) Intel 86/14 computer.

(b) 1/2 Mbyte RAM.

(c) PROM module.

(d) BLC 8222 floppy disk controller module.[1]

(e) 5-1/4 inch floppy disk drive.[1]

(f) Set of COLDBOOT, SIMDOS, and 957B PROMs.

(4) Prolog PROM programmer.[2]

(5) JPL utility software.

V. Transferring IBM PC-Compiled Software

When a program is compiled to run on the IBM PC, a relocatable file, called an EXE file, is generated by the compile and link process. When the program is executed, PC DOS loads the EXE file into memory and then executes it. All input and output to the program is through PC DOS system calls, which are implemented via software interrupt 21H. The utility software performs the functions of locating the EXE file to operate at a specific memory location, loading the file into the target computer's memory, and simulating the interrupt 21H PC DOS system calls. The transferring of the program to the target computer's memory can be accomplished via floppy disk, PROMs, or downloading.

VI. Memory Map of Target Computer

The following is a memory map showing where the various utility programs will be loaded:

FFFF:F Top of memory.

FFFF:0 Jump instruction to the start of COLDBOOT (F800:0).

FC00:0 Start of Intel 957B monitor program (ROM).

FD00:0 ROM image of SIMDOS. Moved to RAM by COLDBOOT during initialization sequence.

F800:0 Start of COLDBOOT (ROM).

8000:0 Start of ROM area containing application program memory image.

Note: The area above 8000:0 is ROM (or unused); the area below is RAM.

1000:0 Start of RAM area where application program is loaded from floppy disk, transferred from ROMs at 8000:0, or downloaded from IBM PC.

07C0:0 Start of RAM resident portion of SIMDOS. Transferred here during the boot-up sequence.

0780:0 Start of the load area for the LOADER utility.

0200:0 Start of RS-232 communication input buffer.

0000:0 Start of interrupt vector table, bottom of memory.

VII. Utility Software

All utility programs were written in "C," or assembly language, with the exception of the Intel 957B program. All programs are small and are easily modified.

INTEL957B. This program is an INTEL monitor program. It is designed to run on the 86/14 computer and to provide basic debugging services. This program is supplied in an unmodified form.

EXE2ABS. This program is written in "C," and runs on the IBM PC. It takes as input a relocatable EXE file generated by the Microsoft compiler/assembler/linker. This program converts the relocatable file into a memory image file designed to run at location 1000:0, although the load address may be changed if desired. The resultant absolute file can be loaded into the target computer via floppy disk using the LOADER utility, or it may be converted to a hex file for PROM programming using the ABS2HEX, and HEX2ROM utilities, or it may be downloaded to the target computer using the IBM2SBC utility. The EXE2ABS places a header at the beginning of the actual program which contains items such as the program length, program start address, a valid file ID mark, a file checksum, and code to initialize the various segment registers, stack pointer, and instruction pointer.

SIMDOS. This program, written in assembly language, runs on the target computer. It is designed to simulate PC DOS functions which the application program calls via software interrupt 21H. A ROM image of SIMDOS is moved into the RAM area during the initialization sequence of the COLDBOOT utility. Once moved, the SIMDOS initialization routine is called, which sets up the interrupt 21H vector.

Since application programs reside in ROM-based assemblies having no disk drives, the majority of the PC DOS functions relating to file and memory management are not needed.

[1] Required if floppy disk loading is desired. While the utility software is presently written for the National BLC 8222, the software can be modified for other controllers.

[2] Required to program PROMs. While the utility software is presently written for the Prolog programmer, it can be modified to control other programmers.

Only the functions required to support the RS-232 interface and a few miscellaneous others are implemented at this time. These include the following:

0 – Program terminate

1 – Keyboard input

2 – Display output

6 – Direct console I/O

7 – Direct console input without echo

8 – Console input without echo

9 – Print string

B – Check standard input status

25 – Set interrupt vector

30 – Set DOS version number

35 – Get interrupt vector

40 – Write to a file or device

44 – I/O control for devices

For a detailed explanation of these functions, consult the PC DOS documentation.

COLDBOOT. This program, written in assembly language, is the first program to run on the target computer following a power on or reset. Upon start-up, COLDBOOT sets the baud rate, and initializes buffers required for the RS-232 I/O channel under interrupt control. Secondly, this program tests RAM from 0000:0 to 8000:0. If during the RAM test time (about 5 s) a key is pressed, a menu will appear following the completion of the RAM test, giving various options to the user. If a key has not been pressed, as is the case in normal operation, the program checks to see if a floppy disk is present and ready to be loaded. If so, the program on the floppy disk is loaded and executed. If no floppy disk is present, COLDBOOT checks to see if a ROM program is present. If a ROM program is present, it is transferred from the ROM area to the RAM area and then executed. If neither floppy disk nor ROM is present, the menu is presented. COLDBOOT always checks for a valid program ID mark, and performs a checksum of the RAM program prior to executing it. The menu items include the following:

(1) Load from a floppy disk.

(2) Transfer a ROM program to the RAM area.

(3) Test RAM.

(4) Execute the 957B monitor.

(5) Perform a checksum of the RAM program.

(6) Execute the RAM program.

(7) Download a program into RAM.

LOADER. This program, written in assembly language, runs on the target computer. It is used to load a program into the target computer via a floppy disk. Using the IBM PC, the floppy disk must first be formatted without the /S option, thereby creating an empty disk. LOADER must be the first program written to the disk. Next, SIMDOS is written to the disk; and finally the application program, after being located to run at 1000:0 by the EXE2ABS program, is written to the disk. Following this procedure, the disk is ready to be loaded into the target computer. This process is not as complicated as it seems if a PC DOS batch command is used.

When the disk is inserted into the floppy drive of the target computer and the reset pressed, COLDBOOT will detect the presence of the floppy disk and load LOADER into address 0780:0, and then jump to 0780:0. The LOADER will then load the disk copy of SIMDOS into its proper place in memory, and then load the application program into location 1000:0. Valid program ID marks are checked, and a checksum of the application program is performed prior to program execution. By using the LOADER utility, a test version of either SIMDOS or the application program can be loaded into the memory of the target computer.

ABS2HEX. This utility program, written in "C," which runs on the IBM PC, converts an absolute file created by the EXE2ABS utility to a hex file which will then be used to program PROMs. ABS2HEX prompts the user for ROM size, and then generates hex files grouped as ROM images. The program also generates and records in the file the checksums for each ROM image.

HEX2ROM. This program, written in "C" and run on the IBM PC, reads a hex file generated by the ABS2HEX utility and then controls the PROM programming process. HEX2ROM is presently designed to control a Prolog PROM programmer but may be modified for others if desired. The user is directed by prompts during the programming process to perform checksums on the programmed ROMs.

VIII. Conclusion

The software approach described above is presently being used to implement the VGTA Software. PCTA software, written in Pascal and compiled using CPM-86/Pascal MT+86, is presently being rewritten in "C" in order to eliminate bugs,

and to make the programs more manageable. Approximately 70% of the programs have been converted and tested. Results show a 20% decrease in code size, and at least a 50% increase in speed. Little assembly language programming has been required in the application programs other than the interrupt linkage routines; this was not the case in the Pascal versions. So far, no errors have been detected with the Microsoft products.

Reference

1. Brown, D. W., Cooper, H. W., Armstrong, J. W., and Kent, S. S., "Parkes-CDSCC Telemetry Array: Equipment Design," *TDA Progress Report 42-85*, Jet Propulsion Laboratory, Pasadena, Calif., pp. 85-110.