

Obsolete Processor Replacement Program

Background

Processor obsolescence is becoming an increasingly vexing problem for embedded computer users. From military and civilian aircraft, to manufacturing facilities, to air traffic control facilities, aging computer systems present a variety of problems. Sometimes the problem is diminishing availability of spare parts for computer repair. At other times the problem is the inability of the processor to absorb new requirements because of limited processor throughput or memory. Regardless how the problem manifests itself, a solution, which is often proposed, is to replace the processor with a more modern piece of equipment. This solution certainly answers the problem with hardware, but leaves the question of what to do with the software which the processor runs.

The natural tendency is to rewrite all software impacted by a hardware change in whatever modern high order language is currently in fashion. Ada has been the favored language for the past several years, with C/C++ gaining favor now. Rewriting software can be quite time consuming, even with language-to-language translating software helping the programmers. Furthermore, recertification of the software for flight safety, nuclear readiness considerations, and other reasons will be required in addition to the normal functional testing. Since many aircraft containing obsolete computers/processors are nearing the end of their service life, the considerable investment to replace computers and rewrite software argues against the improvements.

Fortunately, there is an alternative which may solve this problem for many obsolete processors and their software. Many of the computers being replaced today have software written largely in some high order language like JOVIAL, FORTRAN, or COBOL. By properly selecting the new host processor, compilers may be located to reuse the majority of the older software in the new computer. This process has been used several times by Ball Systems Engineering Operations to port several embedded computer applications to new processors. In fact, the same applications were also ported to ordinary desktop computers. The ability to target many different types of computers with the same basic software gives the program office great flexibility in choosing the replacement computer.

Process/Tool Description

At a very basic level an embedded computer and its software may be considered as a layered entity, as in Figure 1. The physical layer is the actual processor, the wires, and supports for the embedded computer itself and its connection to the rest of the world. The application layer is made up of software units which accomplish calculations and manipulation of data which are input and output over the physical layer. There is an intermediate layer which handles the details of getting data into and out of the physical layer. This hardware abstraction layer hides the complexity of dealing with the hardware from the ap-

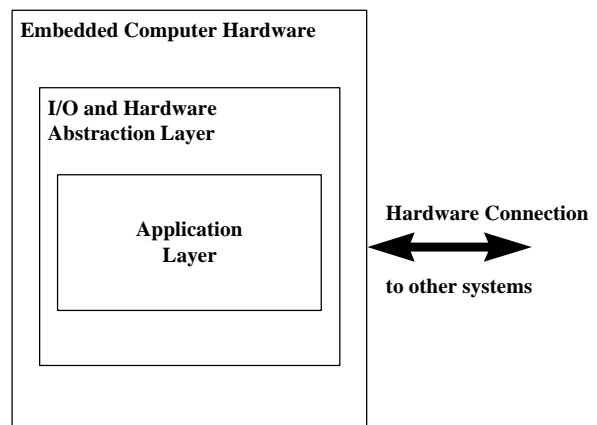


Figure 1 - Embedded Computer Software

plication and its programmer. On a desktop PC, the operating system (DOS, Windows, etc.) provides this intermediate function, allowing application programmers to easily do things such as open data files and write to the screen. Most embedded computer OFPs have this hardware abstraction layer as part of the OFP itself, but implemented in well defined units that can be separated from the application layer. This separability is the key to successfully porting the application to another processor. The process of porting the software can now be defined in three phases: a) replicate hardware functionality, b) rewrite hardware abstraction layer, and c) recompile and link application layer.

The hardware layer of an embedded computer not only provides the actual processor and its associated memory, but also any hardware necessary to drive data lines and busses that connect the processor with the rest of the world. Communication hardware such as MIL-STD-1553B, ARINC 429, RS-422, and discrete signal processing devices must be available for any alternate processor under consideration. Open architecture considerations demand that an open backplane be available so that a given processor may be customized for many different applications by simply installing appropriate memory and communication hardware, along with any special purpose processing boards.

Computer systems built around the Intel family of processors (80x86, Pentium, etc.) are available in many different form factors, including those for PC104 and VME backplane applications. These systems offer relatively low cost and contain little or no proprietary hardware or software, making them extremely desirable targets for rehosted software. Furthermore, communication hardware and other input/output devices are readily available to work with these processors.

The Intel processor family is attractive not only from a hardware perspective, but also from the software development perspective. Compilers, cross-compilers, and assemblers are available for most high order languages (e.g. JOVIAL, Ada, Pascal, FORTRAN) which target these processors. Some modifications may be necessary to allow the embedded computer hardware abstraction routines to address the new hardware. These may be rewritten entirely, or simply adjusted to the new hardware. The new or rewritten routines take the place of the original hardware interface source code.

Once the application layer routines have been separated from all hardware differences between the original and replacement computers, they may usually be easily recompiled to execute on the Intel processor. There are some known potential problem areas, but each has an acceptable solution. For example, word-length and the "little endian/big endian" problems must be addressed. Data structures may need to be packed in a different manner, and pointer math may need to be adjusted. Differences in precision of floating point calculations must be considered. We have been able to easily solve all such problems in the programs rehosted under this philosophy.

Example

We have created several “in-a-PC” emulations of embedded computers. The best example of this is the combination of the Control Display Unit (CDU) and Mission Computer (MC) of the Pave Low helicopter flown by the Air Force Special Operations Command. Figure 2 illustrates the configuration of the system on the helicopter. These embedded processors receive inputs from the navigation subsystems (Inertial Navigation System (INS), Global Positioning System (GPS), and Doppler radar), handle all flight planning and navigation functions, provide flight cues to the aircrew, and provide threat warning and notification. The CDU is the primary crew interface, and also provides a minimal capability “backup mode” of operation in the event of MC failure.

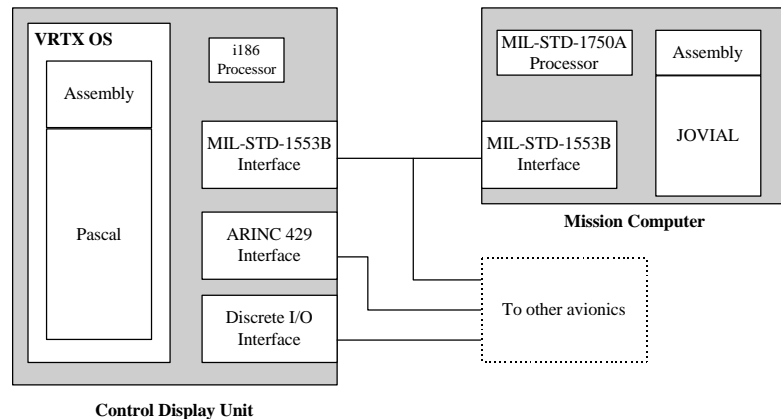


Figure 2 - Pave Low Configuration

The hardware abstraction layer for the CDU is provided by the VRTX operating system and some assembly language routines, while the MC has this capability built into its assembly language executive. The application layer for the CDU is coded in a combination of i186 assembly code and Pascal. The MC software is written in JOVIAL and MIL-STD-1750A assembly code.

The VRTX operating system is a real-time, multi-tasking operating system. The OFP of the CDU is executed as a series of tasks, which are controlled by the operating system. A version of this operating system exists for other Intel processors, including the i386, i486, and Pentium class of processors used in most desktop computers. By outfitting our PC with appropriate communications cards (MIL-STD-1553B, ARINC 429, etc.) and rewriting some of the assembly language routines, the CDU OFP was made to execute in the PC. Since all processing (except that dealing directly with the hardware interfaces) is exactly the same code that executes in the actual embedded computer, the desktop computer operates *exactly* the same as the aircraft system. In fact we have taken this device to the aircraft, removed the real CDU, and plugged our rehosted system into the data bus without the other aircraft computers knowing the difference.

The device outlined above was designed to provide software development and testing capability for the CDU, as well as some operational part-task training capability. However, the CDU only operates in a “back-up mode” if the MC is not on the bus. For the aircrews to use the device to train in primary mode, or for programmers to accomplish complete testing of modified software, a MC was required. We solved this problem by taking the MC OFP through the same “in-a-PC” process. By separating the application layer from the hardware interfaces, we were able to encapsulate the MC OFP as a task under the CDU’s operating system. The resulting training application configuration is illustrated in Figure 3. The system can now be operated in either primary mode (both embedded computer OFPs active) or in backup mode (MC OFP deactivated).

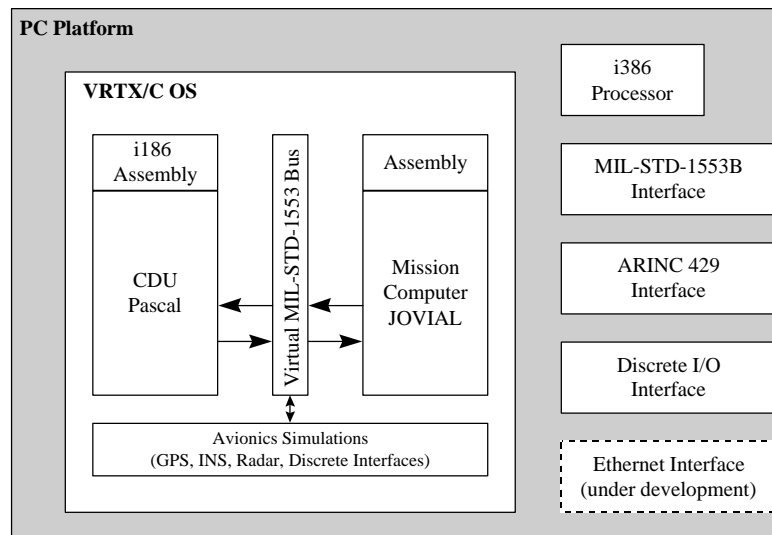


Figure 3 - "in-a-PC" Configuration

To complete the system, we included simulations of several on-board navigation subsystems. The aircraft in question uses inertial navigation units, GPS receivers, and Doppler radar systems as primary navigation aids. Simulations of these systems, along with appropriate flight models, were written and encapsulated as yet another task under the CDU real-time operating system. The increasing power of the Intel processor family allows the two embedded computer OFPs, the operating system shell, and the avionics simulations to execute even faster than on the actual aircraft.

The system configuration of Figure 3 incorporates Pascal, Intel assembly code, JOVIAL, MIL-STD-1750A assembly code, and C source code in one application. Since the application layers of the two embedded computer OFPs are recompiled from the same source that generates the fielded OFPs, the operation of the replacement system is indistinguishable from the original system. Not only can this device be used to train crews to navigate the aircraft, the system could be plugged into the aircraft data busses and used to actually navigate the aircraft.

The Pave Low “in-a-PC” devices form the core of a desktop part-task training system which has been delivered to the 18th Flight Test Squadron at Hurlburt Field, FL, and to the 58th Training Squadron at Kirtland AFB, NM. A similar system has been developed for the AC-130 Gunships, allowing crew station training devices to be developed. These systems are also being used by software engineers as development and test stations for their embedded computer software. Finally, a proposal has been submitted to replace two obsolete processors on the Pave Low helicopter with a single, newer processor that combines the functionality of the older systems.

Limitations

While the process outlined above can prove successful in many applications, it is not a “magic bullet” that will solve all obsolete processor problems. There are also many technical problems, many dealing with the structure of an OFP’s source code or data, which must be addressed.

- **Hardware dependency** – Some applications exhibit a high degree of dependency on specialized hardware. Graphical displays often fall into this category. These systems usually depend upon specialized graphics hardware to manipulate graphics memory and execute graphics primitives (points, lines, circles, etc.). Any rehost effort must reproduce these primitives and memory functions within the application software, or replace the graphics hardware with a functional equivalent. Candidate systems for this rehost process should be analyzed in detail to determine if hardware dependencies will prevent success.
- **Compiler availability** – If the application is written in a language for which there is not an equivalent compiler for the target replacement processor, significant chunks of the application may have to be recoded. For example, some older systems are programmed largely in assembly language. These types of applications are not easily portable to other processors. Cross-assemblers exist for many older commercial processors, but not necessarily for specialized military processors found in many embedded systems. Even applications written in high-order languages may have some level of assembly language routines. There is a “breakeven point” at which an increasing number of lines of assembly code make the rehost process impractical.
- **Data structures** – Most modern processors work on 32 or 64 bit data “words.” Older processors worked on 8 or 16 bit data. This can lead to significant problems with indexing into large data structures, or in locating data structures within processor memory. The “big-endian/little-endian” problem may also present itself, depending upon combinations of old and new processors. Having accomplished this type of rehost many times, Ball SEO engineers can recognize data structures and source code algorithms likely to become problems.

Conclusion

The example above shows that this process has been applied to embedded computer software written in a variety of high-order languages and assembly code. Ball Systems Engineering Operations have accomplished many rehosts of the type described above. Our engineers understand the limitations of the process, and are experienced in identifying likely candidates for the process. Likewise, our engineers are familiar with the technical and structural problems presented by this technology, and with the software adjustments required to solve these problems.

While the rehost process has certain limitations, many applications may be rehosted to newer, more powerful processors with little difficulty. If the new processor is chosen carefully, the problem of processor obsolescence may itself become obsolete.

For more information about these ideas, or to arrange a demonstration of the software systems previously rehosted by Ball System Engineering, please contact Mr. Derek Maddox in our Dayton offices at 937-429-5005, or Mr. Terry Benoit in Warner Robins at 912-922-4363.