

Verifying Modern Processors in Integrated Modular Avionics Systems

Louis Bolduc

AlliedSignal Aerospace, Advanced Systems Technology Group

Louis.Bolduc@AlliedSignal.com

1. Motivation

In the past 30 years, digital computer systems have been incorporated in aircraft, taking on increasingly critical roles in many aspects of flying, such as flight control, navigation and safety.

Federal Aviation Regulations state that systems must be designed so that the occurrence of any single failure condition which would prevent the continued safe flight and landing of the airplane is extremely improbable [1], where *extremely improbable* is quantitatively defined as less than 10^{-9} per flight hour. Qualitatively, such event is not anticipated to occur in the entire fleet operational life. Avionics systems are therefore designed and implemented following strict processes and subject to the most stringent verification to assess compliance with safety objectives.

An important element of the design effort of avionics systems is the attention given to fault containment. If a fault was to occur, it should not trigger the kind of failure condition described previously. Until recently, fault containment was naturally achieved by the physical separation of Line Replaceable Units (LRU). This paradigm is now starting to change with the development of Integrated Modular Avionics (IMA) [2]. Powerful microprocessors make it possible to integrate several functions, such as flight control, radar and guidance, on the same Line Replaceable Module (LRM). Modern Processor features, such as the Memory Management Unit (MMU) and the User/Privileged mode, provide the means to strongly partition the functions spatially and achieve fault containment. Temporal partitioning is also typical of IMA systems, where each function is assigned a fixed predetermined CPU time slice, and is unconditionally preempted at the end of its slice by a hardware timer. In IMA systems, spatial and temporal partitioning are commonly implemented by a small proprietary System Executive (SE).

The close integration of functions on LRMs raises the question of partitioning strength. Are there any conceivable fault scenarios or weaknesses that would allow a function to compromise the operation of potentially flight critical functions? This work focuses on a methodology to uncover weaknesses in IMA hardware / software implementations.

2. Potential Weaknesses

We see at least three areas initially worth investigating: processor faults, spatial faults and temporal faults.

The notion of processor faults is not one to be taken lightly in safety critical systems. Processor faults typically result from bugs in the hardware. For example, consider the Pentium F00F Bug [3]. When P5, P54 or P55 processors encounter the instruction F0 0F C7 C8, or anything from F0 0F C7 C8..CF, the processors lock up, regardless of their mode of operation. Identifying those processor versions that are unsuitable for critical airborne systems is one of our objectives.

Spatial faults refer to a breach of the functional partitioning, where one or more functions become contaminated by the actions of another function. Theoretically, the SE should properly configure the MMU and User/Privileged mode to prevent such occurrences. Temporal faults have similar properties. They are characterized by a disturbance in the time slice allocation as a result of the actions of one function. In both cases, a processor fault or SE design or implementation faults could be at the root of the weakness. Processor and SE faults are the initial focus of this work.

3. Verification Method - Processor Faults

The verification method we are exploring aims at uncovering both processor and System Executive faults. The nature of the faults we are looking for directly influences the method. Let us take once again a closer look at processor faults.

Few processors implement an actual executable instruction for every bit pattern that can be fetched. The F0 0F C7 C8 pattern described earlier is not documented as an executable instruction. In some cases, undocumented instruction patterns do implement actual instructions that the manufacturer has chosen not to disclose. They may be part of a design legacy or part of a roadmap for future versions. Typically, processor manufacturers will qualify instructions as defined, illegal or reserved.

Undefined, illegal and reserved instructions are the primary focus of this work, but defined instructions are

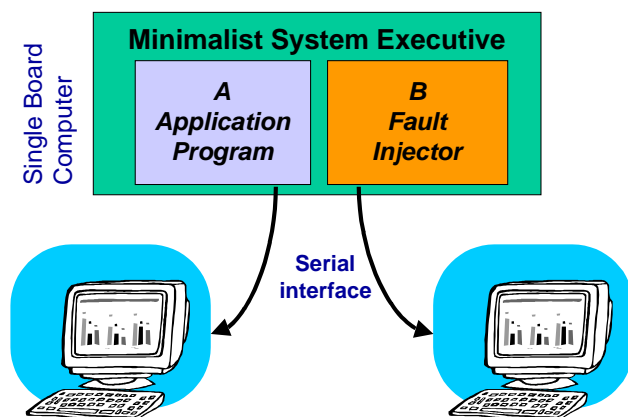
not to be completely left out either. The PowerPC for instance, has defined instructions whose results can be *boundedly undefined* under certain circumstances. Anecdotal reports of faults similar to the Pentium F00F Bug, but involving legal instructions on the 80386 and Sparc SS5/170 have also been published on the Internet [4]. Those reports will be examined to determine whether there are grounds to increase the scope of this research.

Our approach to uncovering processor faults is both analytical and experimental. Initially, publicly available documentation on the processor of interest is examined to find out known errata and features of the processor design that might lead to a deadlock, undefined results or might be considered weaknesses in the processor design. Since our aim is not to try to identify all faults, but to determine whether a particular processor is suitable for IMA, we stop as soon as one fault has been uncovered. The analysis of documentation may be sufficient to meet this condition. Note that processor features yielding undefined results may not constitute grounds to declare a particular processor unfit, but may direct our attention in the experimental part of our method.

The experimental part of our method involves the use of a test vehicle. This test vehicle consists in a minimalist System Executive that embodies the generally accepted architectural features of IMA and can be targeted with relative ease to any modern processor that we wish to test.

The test vehicle runs two processes: a simple fault-free self-testing application, and another process that acts as a fault injector. It comprises mainly of illegal, reserved and undefined instructions for the processor under consideration. Each application outputs information over an RS-232 connection, providing health information in the case of Process A, and exception messages in the case of Process B.

Process A exercises all user level registers (integer and floating point), monitors its mode of operation (User vs. Privileged) and the duration of its time slice. It also



exercises as many user-level instructions as possible in an attempt to detect any anomaly in its operation. Process B monitors whether program exceptions are indeed occurring when illegal, undefined or reserved instructions execution are attempted. A change in privilege level is also monitored. The minimalist System Executive monitors privileged registers for unexpected change.

The violation of any preset rule, such as a change in privilege level, privileged register, absence of exception following the attempted execution of an illegal, undefined or reserved instruction is analyzed and may be sufficient to declare the processor unfit for IMA.

4. Verification Method - System Executive

The exercise of developing and targeting our test vehicle for specific processors enables us to appreciate their idiosyncrasies and potential weaknesses. This knowledge is then used in the crafting of test cases for actual IMA System Executives destined for these processors.

As an example, the PowerPC Decrementer exception is a natural choice to implement the CPU preemption, but has a lower priority than the External interrupt exception. This suggests a test case where an endless loop in the External Interrupt exception handler would cause a system failure if the IMA System Executive designer had not taken this into consideration.

The test cases are collected. Over time, they will form a general test suite for IMA, comprising of processor independent an processor specific test cases.

5. Roadmap and Project Status

Our intentions are to investigate and identify sources of common software failures and weaknesses in modern processors, and generalize the approach into an overall methodology for IMA systems.

At the beginning of September 1999, our test vehicle had been implemented and was running on a PowerPC 603e processor. Our attention is now shifting on the development of the two processes, the self-testing application and the application consisting mainly of illegal, reserved and undefined instructions.

References

- [1] FAR 25.1309 (Amendment 25-23)
- [2] Design Guidance for IMA, ARINC Report 651-1
- [3] Dr. Dobb's Journal, May 1998
- [4] www.geog.ubc.ca/snag/bugtraq/msg00857.html