

Techniques for achieving reliability in safety PLC embedded software

Dr. William M. Goble
www.exida.com

ABSTRACT

There is a strong trend toward the use of programmable electronics in safety instrumented systems. Yet some users still avoid software-based systems. They cite the unpredictability of software and case histories of software failure. However, a special class of PLC called a “safety PLC” does meet the need for safety and high availability in critical automation.

A safety PLC must meet the requirements of a set of rigorous international standards that cover the design, the design methods and testing of software and hardware. Third party experts (typically TÜV in GERMANY) enforce the rigor when the products go through the certification process. Some of the methods used to build “high integrity software” for safety PLCs are described in this paper.

INTRODUCTION

The quantity of software in equipment used for critical process control and safety instrumented systems is growing. This is due to a strong trend toward using flexible safety PLCs instead of relays or DCSs in safety instrumented systems. Safety PLCs are microcomputer-based controllers that are designed for high safety and high availability applications. Safety PLCs offer application flexibility, self-diagnostics, communication interfaces to other plant automation systems, automated application tools that help prevent human error [1] and a level of reliability and safety not available in conventional PLC/DCS equipment.

A PLC qualifies to be called a safety PLC when it passes a series of tests given by third party certification agencies (TÜV, Germany or FMRC, US). Safety PLCs are certified per international standards, primarily IEC61508 [2] and VDE0801/A1[3]. These standards require extensive safety analysis of both hardware and software. A key part of the analysis covers the diagnostic ability of the PLC. In the VDE0801/A1 standard, the qualitative rule “no known dangerous undetected failures” applies. In the IEC61508 standard, detailed quantitative analysis [4,5] of hardware failures must be performed. That analysis determines the “diagnostic coverage factor,” a number between 0% and 100%. Levels of 90%+ are expected, depending on target safety integrity level and amount of safety redundancy. The safety PLCs are also evaluated to insure electrical safety, user manual integrity, fault tolerant architecture

and software integrity. The software integrity is another of the key differences between conventional PLC/DCS equipment and safety PLCs.

HIGH INTEGRITY SOFTWARE

While some regulatory bodies in certain geographic areas still do not allow software-based equipment to be used in critical process control or safety protection applications, most have recognized the value of the intensive diagnostics available in safety-certified software-based controllers. Those regulators who do not allow software cite the unpredictability of complex software and the history of software failures [6].

There may be reason to doubt the reliability and safety of some types of consumer grade software, but the international standards used by designers of safety PLCs have rigorous requirements to increase software integrity. The standards emphasize the process: product development according to a lifecycle model. While several models are available, the “V-model” is the recommended choice because of the link between the design and test specifications during product development. (see Figure 1) Software techniques for complying with these requirements will be discussed later.

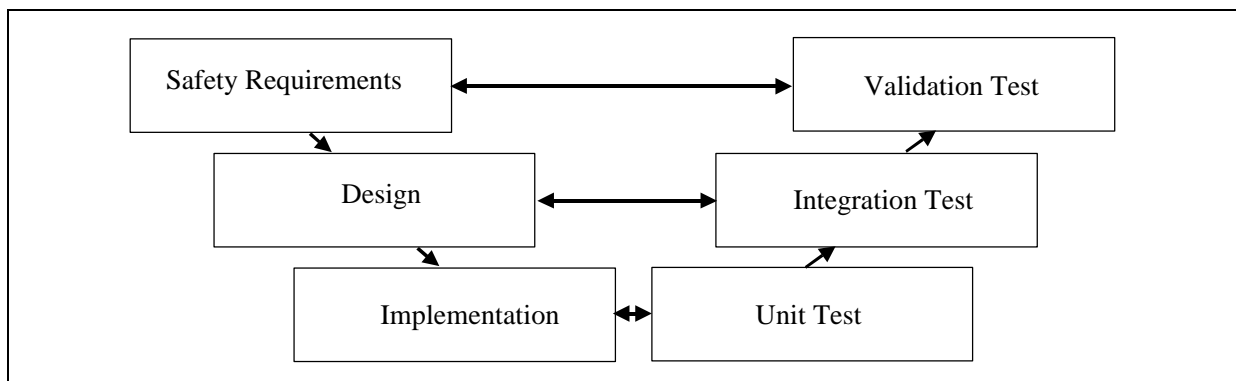


Figure 1: V-Model, Software Development Process

The standards cover the entire development process from functional requirements of the product to final testing, not just software implementation. International standards require a whole set of development activities designed to insure the highest software quality for avoidance and control of faults. These activities include program execution diagnostics, data verification testing, data storage integrity, complexity reduction, and a wide set of software development process requirements. Following these guidelines closely with the certification agency’s help will result in “high integrity software.”

Techniques for achieving reliability in safety PLC embedded software

Overall, the safety standards require a quality and robustness not found in many types of products, with or without software. Whether the VDE0801/A1 rules or the IEC61508 rules are being applied, they both dictate a more stringent product development effort. The software development of these products must include many techniques that might be cost prohibitive (in both time and money) to average software suppliers.

CRITICAL SOFTWARE PROCESS

Quality principles developed by Juran and Deming are well known throughout the world for factory operations. These quality principles require that a process be established and followed. While following a process may seem obvious, it is easy to take software quality for granted and shortcut the process after the initial design is completed. This seems to be part of the “software culture” at times, especially when a project gets behind schedule.

The safety critical software development process emphasizes a V-model that starts with product requirements. Requirements reviews determine that all safety relevant requirements are documented. As the V-model indicates, product validation tests are developed along with product requirements. Test planning can and should be done while requirements are being finalized. A test plan review provides a good crosscheck of the testability of any given requirement, a test of requirement reasonability. The test plan review may also uncover missing requirements before too much design has occurred.

The requirements are considered the foundation of the whole project and as such should be treated quite seriously. Each requirement must state the safety function in quantifiable terms (“The analog channel shall detect any faults that cause a value greater than +/- 2% of span within one second”). An important aspect of the process is the traceability of requirements to tests. While this step makes auditing easier, it also aids the developers to identify missing and duplicated requirements. The test effort must show correctness and completeness of fulfilling the product requirements. Correctness means that the software operation performs exactly as it is intended, fulfills the matched requirement, and takes appropriate action for fault detections. Completeness means that all requirements have been met.

MANAGE THE CHANGES

It is essential for the development team to maintain control over changing requirements. Documents should be properly identified and include revision history. Formal reviews should be held with meeting minutes that include issue resolution and agreed action items. If decisions are made that affect requirements, the team must go back through the process and judge impact to other parts of the product. The project manager must review and assure completion of all action items. More

importantly, the team must translate informal resolutions of design issues to the design documents. Not every design decision is made by a formal review; many decisions can and should be made at the level appropriate for implementing the decision. When decisions are made in this manner, the appropriate design documents should be updated. The document trail serves to inform all project stakeholders of the changes.

SAFETY PLC SOFTWARE TECHNIQUES

Failures in software do not occur randomly nor does software “wear out”; all software failures are designed into the system. When that certain combination of inputs, timing or data presents the right conditions to the system, it will fail every time. For this reason, failures in software systems are known as “systematic” failures. To make certain the software is performing as intended therefore, the software must check itself to make sure it has what it thinks must be done. Software diagnostics are programmed into embedded code. One of the most effective software diagnostics is “flow control.” Program flow checking makes sure essential functions execute in the correct sequence. At key points in the program, a “flag” is set, preferably with a time stamp (Figure 2). At the end of each program scan the flags are checked. All flags must be set in the correct sequence. If time stamps are also used, the time difference between flag settings can be compared with reference values for further error detection.

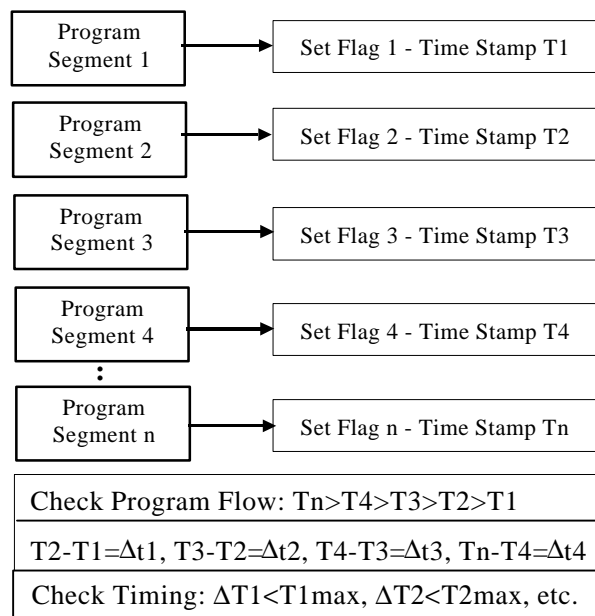


Figure 2: Program Flow Control

Another software diagnostic is called “reasonableness checking.” When the results of computations should always be within known limits, the computed outputs can be tested to see if they exceed those known limits. In this way systematic faults can be detected before an erroneous system action occurs. Aside from computational results, many states and values are derived and stored within software control. When values are mutually exclusive, additional reasonableness checks on this data can flag faults before erroneous states occur. The same mechanism can be used for message schemes between software-based systems.

The data used in a safety PLC must be protected from corruption. Critical data is identified by analyzing the execution flow of critical software functions. Often done with dataflow diagrams, this analysis identifies the software processes that perform critical functions found in the safety requirements. These functions include both the diagnostics and the execution of the user safety program. The data associated with these software processes is termed critical data. Critical data must be stored in a manner that cannot become corrupted in an undetected manner by systematic software fault or by hardware failure.

Figure 3 shows a dataflow diagram with a chain of processes and a reverse calculation check on critical data. Process #8 provides a crosscheck on processes #1 through #3 to detect an error in the normal process chain. While processes #1-#3 may provide a high accuracy result based on product specifications, process #8 provides a comparison of that result within the product safety accuracy, which is usually less accurate but will detect an erroneous software condition.

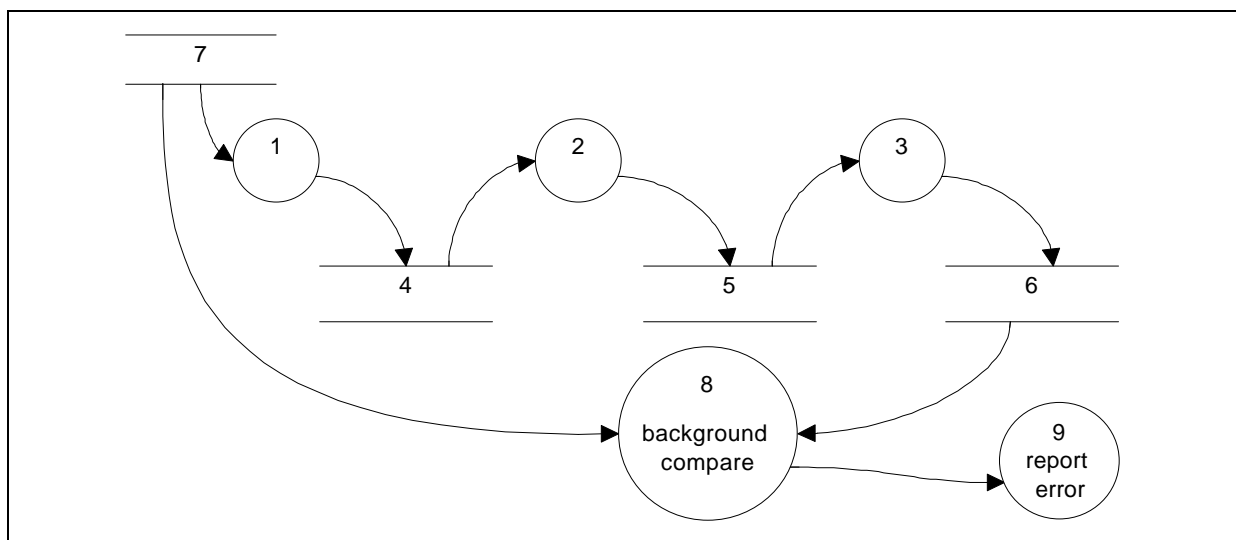


Figure 3: Dataflow Diagram With Reverse Calculation Comparison

FIREWALLS AROUND CRITICAL FUNCTIONS

When safety critical functions must be combined with non-safety critical functions, the design must include sufficient safeguards for non-interference. This means that any non-safety operations, like data acquisition from a safety system to a plant manager console screen, cannot hamper or inhibit in any way the safe operation or fault detection mechanisms of the safety system. If any non-safety functions have the possibility of writing data to a safety system, the writes must be under controlled circumstances in an allowed configuration mode. The system design must reject any unexpected changes to the system.

SOFTWARE COMPLEXITY

Safety PLC standards demand special techniques to reduce software complexity. Operating systems are carefully examined for task interaction. Real-time interaction, such as multitasking and interrupts, are avoided. This is because many of the most insidious software faults have been traced to unanticipated interaction between software programs and common resources used by multiple software tasks. When multi-tasking is used, real time interaction of tasks requires extensive review and testing. It is especially important to avoid the use of common resources, such as I/O registers and memory, by asynchronous tasks in a multi-tasking environment.

TESTING

Extra software testing techniques are required for safety PLCs during software development. The findings and assumptions of the criticality analysis must be proven. A series of "software fault injection" tests must be run to verify data integrity checking. The programs are deliberately corrupted during testing to insure predictable, safe response of the software. Hardware emulators, specific for the microprocessor, are often used to set break points and alter program data, then the program is allowed to continue to see if the fault was detected. An alternative test method uses custom software built into the program. This requires a monitor program to accept user input about special test codes. These test codes invoke fault injection functions that are time dependent and not easily performed by an emulator. The testing must be fully documented such that third-party inspectors can understand the operation. While this activity is not justified in most software development, this is exactly how the most harmful and covert software design faults are uncovered.

FAULT AND CHANGE TRACKING

When suspected problems are found in the software design or code, they must be recorded and reviewed using a formal system [8]. Not every reported problem is a real defect, and these should be discarded with rationale for the determination. Not every problem found is reliability or safety related. When a problem is investigated

and deemed important enough to fix, the development team should perform an impact analysis of the suspected defect. The analysis should include:

- *Accurate problem description*
- *Effect of the problem on critical functions*
- *Description of the proposed solution*
- *Effect of the proposed solution on safety functions*

A database should contain all necessary details of activity related to problem identification and tracking. Items to clearly identify in this database are:

- *Author, date, and product/version where problem was found*
- *Problem description, with any particular test setup details or circumstances*
- *Implementer log that includes change notes and files affected*
- *Authorization notes for accepting the change*
- *Time estimates and actual time used*
- *Test data to see that the fix was correct*

SOFTWARE PROCESS IMPROVEMENT

Problems discovered in the software development process that involve safety critical functions must be treated with great scrutiny. The step in the development process where the problem occurred should be identified [9]. Some problems can be traced to design or implementation, but the greater number of problems is often traced to missing or inadequately defined requirements. When the latter case occurs, the lifecycle model loop must be reviewed to determine where to start implementation of the fix and any related documents that need to change.

It is also useful to identify what error detection step in the development process should have found the problem. If the problem was discovered at a later step in the process, improve the process for future developments [9]. While it sometimes seems like a problem is isolated to a specific area of software, it is often the case that the problem is more far-reaching. The design documentation referenced by the problem area must be reviewed for non-obvious interface effects. For example, there can be subtle timing elements that could affect message schemes that are safety critical, or an uncommon but likely mode of operation may inhibit a critical diagnostic under specific conditions.

Any quality control effort's goal is to find the root cause and fix the process in an irreversible way. An effective problem tracking system will aid in closing the loop on problem solving that includes both internal process improvement and field failure analysis. The system can serve as the repository of all investigative findings and include resolution details.

CONCLUSION

International standards for safety PLC software design require an excellent software development process and special software design and test techniques. These techniques will produce more reliable software according to the group of international experts on these standards committees. A PLC that meets these standards provides value through high safety and high availability in fault tolerant programmable systems. A PLC that meets these standards should be approved by regulators for the appropriate safety level to which it was approved.

REFERENCES

1. Goble, W. M. "Meeting Safety Standards with Matrix Programming," *Proceedings of the Automation Exhibition*, ISA Cincinnati, OH: Cincinnati, 1999.
2. IEC61508, Functional Safety of electrical / electronic / programmable electronic safety-related systems, International Electrotechnical Commission, Switzerland: Geneva, 1998.
3. DIN V VDE 0801 A1, Grundsätze für Rechner in Systemen mit Sicherheitsaufgaben, Änderung A1, 1994.
4. Goble, W. M., Bukowski, J. V. and Brombacher. A. C., "How diagnostic coverage improves safety in programmable electronic systems," , " *ISA Transactions*, Vol. 36, No. 4, The Netherlands: Amsterdam, Elsevier Science B. V. , 1998.
5. Goble, W.M., *Control System Safety Evaluation and Reliability*, ISA, Raleigh, N.C., 1998.
6. Leveson, N. G., *Safeware – System Safety and Computers*, Addison-Wesley, MA: Reading, 1995.
7. Lawrence, J.D., and Preckshot, G.G. "Design Factors for Safety-Critical Software." (Report # NUREG/CR-6294) Lawrence Livermore National Laboratory, 1994.
8. Mavis, S. A., "An Organized Way of Tracking Faults in the Development Process," *Proceedings of the International Symposium of Engineered Software Systems (ISESS) Symposium, Malvern, PA, USA, May 1993*, UK: London, World Scientific, 1993.
9. Bukowski, J. V., and Goble, W. M., "Software – reliability feedback: A physics of failure approach," *1992 Proceedings of the Annual Reliability and Maintainability Symposium*, NY: New York, IEEE, 1992.