SOFTWARE-IMPLEMENTED SAFETY LOGIC

Angela E. Summers, Ph.D., P.E., President, SIS-TECH Solutions, LP

"Software-Implemented Safety Logic," Loss Prevention Symposium, American Institute of Chemical Engineers, Houston, Texas, April 2001.
"Software Implemented Safety Logic," Process Safety Progress, June 2002.

Abstract

The process industry relies on various safeguards to minimize the potential and consequence of hazardous incidents. Active safeguards, such as safety instrumented systems (SIS), fire and gas systems, and emergency vents, use process sensors to detect process upsets then take action to bring the process to a safe state.

Many Users implement active safeguards using programmable logic controllers (PLCs). The PLC uses a combination of hardware and software to respond to inputs (process variables) by generating outputs (safety actions). The execution of the safe state logic is dependent on the application program integrity, which may be developed by the PES vendor, engineering contractor, system integrator, or User. Successful implementation and thorough verification of the application program is essential for correct SIS operation.

This paper will discuss the various types of software, including full variability, limited variability, and fixed programming languages. It will then discuss the proposed application program requirements and guidelines in the international process industry sector standard, draft IEC 61511.

Basic Software Languages

There are three software languages used for the programmable devices in safety instrumented systems (SIS): fixed program, limited variability, and full variability. These languages exist at various levels in field devices and in logic solvers.

Fixed program languages are used in many proprietary, dedicated function devices, such as transmitters, smart positioners, turbo-machinery monitors, vendor-packaged fire and gas systems. These devices only allow the user to adjust parameters associated with the device operation, such as the range of the transmitter. The devices do not allow the alteration of the function of the device by the User. Thus, the language at the User level is fixed. The Vendor provides data sheets, user manuals, or technical guides to the User to describe how the device functions and what parameters can be adjusted to configure the device for the specific application. Fixed program language devices have typically undergone extensive testing at alpha and beta level to ensure the device performs as intended.

Limited variability languages are more flexible than fixed program languages. These languages consist of pre-developed and tested functions. Limited variability languages include ladder logic, function block diagrams, and sequential function charts. These languages allow the User to combine the functions to configure or program the required application logic. The Vendor should provide a User manual for the software. This manual should define the basic functional components contained in the software library. This manual should also discuss how to utilize the engineering tool to configure these basic components into an application program.

Full variability languages are general purpose programming languages, such as C++ and Pascal. This type of language uses an operating system, which provides system resource allocation and a real-time multi-programming environment. These languages can be used to create an application program with full flexibility in how the logic is constructed. The use of these languages is generally limited to computer specialists who generate unique coding, such as simulating process operations. For safety instrumented systems, full variability languages are found in embedded software that controls logic solver at the operating system level. These languages are highly flexible, allowing the individual programmer to create different coding each time to execute a specific application.

The international safety instrumented system standard for the process sector, draft IEC 61511, discusses the use of software in the process industry. This standard recommends the use of limited variability languages for application programming. The standard does not prohibit the use of full variability languages. The standard requires that application programs that are created using full variability languages are developed according to IEC 61508, which is much more stringent in terms of application development, documentation, review, and verification than draft IEC 61511.

The following sections discuss in general the philosophy presented in IEC 61511 for developing application programs. Consequently, the discussion is focused on the development of application programs using limited variability languages.

Software selection

It is important to not treat the engineering tool or software as something that simply comes with the logic solver like a computer program on your office PC. The functionality of your application program lives and dies by its code and that code is dependent on two things: errors in the software used to create the application program and errors made in the application program. With so much Vendor software available, Users must assess the software to determine whether it is consistent with the hardware platform, the anticipated size and complexity of the application program, and the skill of the anticipated programmer.

The application program is constructed using the software engineering tool and function library provided with the logic solver. An error free application program can only be developed if the engineering tool and function library is error free. Consequently, the software engineering tool and function library must be fully tested by the Vendor and, preferably, by an independent person or organization to provide the User with a high degree of confidence in the software reliability. The functions in the function library will be used over and over in many potential application programs. An error in the function can behave like a virus by corrupting multiple programs with incorrect logic. Thus, any functions developed by the User using the Vendor engineering tool should be tested not only by the developer but also by an independent person or organization. To facilitate this testing, the software should include tools that aid in verification and testing to ensure that the program executes the safety logic correctly and is of high quality and integrity.

Software should be selected that assists in the control of unauthorized changes to the application program. Consequently, the software selected must have version management capability. Users should show preference for software that has the capability of tracking modifications to a specific function level. Since an application program is only as good as its verification, the software should undergo emulation and/or simulation to facilitate thorough testing of the application prior to use.

Organization

Safety instrumented system application programs execute logic that is directed at preventing or mitigating hazardous events. In creating an application program, the organization of the program is very important. The program should be developed in a modular manner that allows isolation of specific safety function logic. Safety and non-safety programs should be separated from each other and labeled as safety and non-safety. The program should be readable and understandable with enough comments that a reviewer of the program can locate specific logic for modification or verification.

Application program documentation

Application program documentation must be readable and accessible. It must contain at a minimum the following information:

- Application program description
- Legal entity
- Logic conventions
- Library function description
- Order of logic processing of data
- Identification of non-safety functions
- Identification of program elements not used
- Revision control – configuration management

Hard copies of the full application program are not necessary, but the information listed above should be keep readily accessible to anyone who desires to modify the program.

Application program storage

Software never wears out, because it is not susceptible to environmental conditions, process conditions, or external stresses. The software tools and data storage media do have a limited technological life. As computer technology has evolved, data storage media has exhibited a useful life of less than 10 years. After all, 3.5" disks became widely available only 10 years ago and are now relegated to handing files back and forth when e-mail is not available. The SIS application program must be accessible as technology evolves. This means that the User must consider migration paths for the application program and software storage. Finally, back-up. Back-up. Back-up. Keep electronic copies of the as-installed program and the previous version in a safe and locked location.

Conformance with safety requirements

The application program is created to follow the safety function logic described in the safety requirements specification. In order to determine that the programmer has been successful in doing this, a thorough functional test of the application program logic must be performed. Due to the large number of possible combinations of operating and fault conditions, it is impossible to completely test every potential failure path. Therefore, conformance with the safety requirements demands three efforts: understanding design intent, review of program by an independent person, and verification of the program performance.

First, the application programmer must understand the safety function design intent. Incorrect interpretation can result in improper sequencing execution, improper shutdown actions, and conflicting logic. Further, the programmer should include fault detection for field instrumentation, such as out-of-range, drifting, or stuck measurements, alarming these for repair initiation. Also, the programmer must understand SIS resets and start-up permissives. If these are not done correctly, the plant will be difficult to re-start after shutdown. Finally, on-line test logic should be structured to minimize the safety function loss during testing.

Second, a review the application program structure and content should be conducted by an independent person. A functional review can be conducted by a representative of the User organization for whom the SIS is being designed. The User should verify that the logic written is the logic that they intended. This means reviewing the application program at a detailed level, rather than just by looking at the program execution or program description.

Third, a thorough input to output verification of the program performance covering the following areas:

- Safety function logic
- On-line field device testing logic
- Bypass logic
- Deviation alarm logic
- Data boundary tests
- Execution times
- Sequence implementation

The tests can be conducted using simulators or emulators, but at some point, the application program must be tested in the actual hardware configuration, including any peripheral devices, such as sequence of events recorders, communication modules, and remote I/O communication. Remember that there is really no such thing as too much testing. There is simply a point where you cannot test all failure paths.

Test documentation must be completed prior to release of the system for process plant implementation. The test documentation must include the following:

- Test parameters,
- Faults found,
- Fault impact, and
- Fault resolution.

These documentation requirements may seem excessive, but the application program is the backbone of the SIS. If the application program contains faults, the SIS will fail. Hardware failure is dependent on the random failure of components. SIS designers use known failure rates, redundancy, testing, and diagnostics to create high integrity systems. In contrast, application program failure is highly dependent on the engineering tool and programmer, whose failure rates are uncertain. The only way to ferret out these potential failures is thorough testing and documentation.

Modification

The application program should rarely be changed and, when it is, it should be done under management of change. In fact, changing the application program should be treated as seriously as modifying the process. While changing the program may be fast and inexpensive, the change can have a profound impact on plant safety. Due to the relative ease of making program changes, access to the application program must be controlled with password restrictions. The operator should never have the authority or the capability to modify the application program. And, no one should have the capability to over-ride or bypass the application program without management of change approval.

When changes are necessary, the User should evaluate the modification in light of the safety requirements specification. An impact analysis should be performed to ensure that the change does not impact any other safety function. All changes must be recorded in a written log to allow auditing of the program version against the log of approved changes. Unless the change can be sufficiently isolated to a specific program module, the entire program will require re-verification. If isolation is possible, only the affected program module must be tested.

Conclusion

Programmable electronic systems are used worldwide to execute safety functions in the process industries. Their performance is highly dependent on the integrity of the application program. While there are concerns about software reliability, the incredible flexibility and opportunity to create programs with diagnostic comparisons, trend analysis, reactive calculations, etc., make software based systems an excellent choice for safety instrumented systems. However, it is only through careful development, review and management of change that is possible to ensure, with reasonable confidence, that the program is correct.

References

- "Setting the Standard for Safety Instrumented Systems," Chemical Engineering, December 2000.
- "Application of Safety Instrumented Systems for the Process Industries," ANSI/ISA-84.01-1996, ISA, Research Triangle Park, NC, 1996.
- "Functional Safety: Safety Instrumented Systems for the process industry sector: Part 1 Framework, definitions, system, hardware, and software requirements," IEC 61511-1, International Electrotechnical Commission, FDIS (2002).