SIL3 Graphic Integrated Development Environment for a Safe System-on-Chip

EMIL DELIC, KAROLIN LÖSER, MICHAEL SCHREIBER, ALI HAYEK, JOSEF BÖRCSÖK
Institute for Computer Architecture and System Programming
University of Kassel
Wilhelmshöher Allee 71-73, 34121 Kassel
GERMANY
emil.delic@uni-kassel.de, karolin.loeser@uni-kassel.de, michael.schreiber@uni-kassel.de, ali.hayek@uni-kassel.de
http://www.uni-kassel.de/eecs/rs

Abstract: - In this paper a novel approach for development of SIL3 applications is presented. Together with our industry partner we have developed a safe system-on-chip including a safe operating system and a SIL3 middleware controlling all low level access to the hardware. Additionally, graphical programming in application development is provided by a SIL3 graphical integrated development environment which is the focus of this paper. The implementation and features of the SIL3 graphic IDE are introduced. The communication between SIL3 graphic IDE and the safe system-on-chip is described including the use of a “black channel” approach for safe communication. Finally, two simple example applications are given and depicted.


1 Introduction
Safety-related applications are getting more and more complex, while at the same time (software) engineers are urged to take less and less time for development and testing in order to reduce costs of the final product. As safety itself must not be neglected, and developers of safety-related software are supposed to implement safety requirements, e.g. defined in norm IEC 61508-3 [1], new ways to reduce effort in application development need to be found.

The norm IEC 61508 classifies different safety integrity levels (SIL): from SIL1 (lowest level) to SIL4 (highest level). It also provides guidelines on what can be done in order to achieve the aimed SIL. For software development, one recommendation for SIL3 (but not for SIL1) is using formal methods to avoid systematic faults. The effort of implementing new code for an application using these recommended measures and techniques increases with increasing complexity of the software.

Often, safety-software products, which aimed for a SIL certification, were denied this certification. Lloyd and Reeve found the following reasons for the denial [2]:

- Missing requirements specification
- Interaction between hardware and software is not described well
- Missing detailed descriptions and results from static code analysis
- Poor integration and unit testing
- Writing acceptance test specification in the later stages of the software development

In order to reduce the impact of the certification issue, a versatile approach offering different levels of support in application development is presented. Fig. 1 shows the different stages of access to the target platform. The lowest level is the SIL3 certified target platform without any additional software [3]. On top of this, a safety-related SIL3 operating system is placed which takes care of basic settings of the target platform, e.g. setting system clock, handling errors and bringing the target platform into safe state on the occurrence of errors. The next level is the SIL3 certified middleware,
which handles all low-level access to the target platform \cite{4}\cite{5} for the application. The next level is the graphic integrated development environment (IDE), which also will be certified for SIL3 and is the focal point of this paper.

A developer might directly access the target platform on low-level, which demands detailed knowledge in safety-related programming and all requirements necessary to comply with the norm. This approach leaves all possibilities but also all risks and efforts, including certification of all software parts, to the software engineer.

Therefore, the middleware with underlying operation system is intended to take care of most safety-related aspects and to enable the software engineer to write safety-related software nearly as straightforward as non-safe applications. Thus, the focus in development may be set on functional issues, as the safety aspects are covered by the middleware as far as possible. Still, this approach needs detailed programming skills in C++ or C.

Software engineers developing applications for programmable logic controllers (PLCs) may be more familiar with graphic programming languages, e.g. Ladder Diagram and Function Block Diagram. Therefore, a SIL3 certified graphic integrated development environment (IDE) is available, so applications for the target platform may also be created using a graphic IDE. Thus, using the whole package, from the target platform up-to the graphic IDE, application development is enormously sped up. A safety application still needs to be developed in conformance with appropriate norms, but the effort for the certification process would be minimized, as most safety-related parts would already be SIL3 certified. This paper will describe the SIL3 graphic IDE developed by our industry partner. The graphic IDE will be also SIL3 certified. Our contribution is adding support for the IDE to the SIL3 middleware.

2 SIL3 System-on-Chip

2.1 Multi-core Architecture

The target platform is a SIL3 certified multi-core system-on-chip (SoC) architecture which integrates all features of a programmable logic controller (PLC) on a single chip. This reduces the number of required components for safety applications, thus improving system reliability. A more detailed description of this architecture, which has been developed in cooperation between Uni Kassel and our industry partner, can be found in \cite{3}.

The multi-core architecture consists of two subsystems: a safe 1oo2D subsystem (SAFE system) and a non-safe single-core system intended for communication (COM system). The two subsystems are connected interference-free via a dual-ported RAM (DPRAM). In addition, both processor systems may trigger an interrupt in the other subsystem. Both subsystems contain 8051-based processor cores which provide data (SRAM) and program (flash) memories, diverse communication interfaces, timers and interrupt controllers and on-chip debugger units. In addition, the SAFE system features safe digital in- and outputs, counter inputs, PWM outputs and on-chip safety-features. The non-safe COM system can act as black channel for safe communication between SAFE system and field devices for safety-related applications utilizing its Ethernet, CAN, UART or SPI interfaces. In the black channel approach it is up to the communication end-points to implement the safety layer \cite{1}, the operating system and safe system already provides one safe communication.

2.2 Safety-Related Operating System

The first software level on SAFE system is a safety-related operating system (SafeOS), developed by our industry partner, which will be certified SIL3. On COM system a non-safe operating system (ComOS) is running.

SafeOS executes safety-related hardware tests at start-up and during runtime. After initialization, SafeOS provides a fixed cycle time. All tasks are executed sequentially within one cycle; multi-tasking is not possible. If cycle time is exceeded, or an error occurs, SafeOS initiates the safe state for
the target platform. SafeOS conducts the following functions:

- Monitor fixed cycle time
- Test SAFE system
- Provide safe data for system communication
- Provide safe protocol stack

ComOS has no fixed cycle time. All pending tasks are executed sequentially as fast as possible. Then, the next cycle starts. ComOS conducts the following functions:

- Configuration of Ethernet switch
- System communication via Ethernet and CAN
- Diagnosis (during development phase)

### 2.3 SIL3 Middleware

The SIL3 middleware comprises all safety-related functionality of the target platform and all low-level access is encapsulated by the middleware. The middleware is built up in modules and each independent module controls a single functionality of the target platform. Thus, the modules represent low-level drivers for all safety-related functionalities. A detailed treatise of the concept and validation of the SIL3 middleware is presented in [4][5]. In order to save resources, a library contains all modules, made using IAR XLIB Librarian and XAR Library Builder [6]. The required program memory is kept as small as possible by only linking modules which are required by the application. As the middleware is designed to be used on an 8-bit, 8051-based CPU core, this is essential.

The middleware includes the following low-level drivers:

- IOCDevice for safe digital input and output ports
- PWMDevice for safe pulse-width modulation outputs
- CounterDevice for safe counter inputs
- UARTDevice for UART communication
- SPIDevice for safe SPI communication
- CRCDevice for cyclic redundancy check (CRC) calculations
- DPRAMDevice for control of DPRAM (communication with COM system)

The middleware is called in each SafeOS cycle. At first call, instances of all modules, used by the application, are created. Only one instance is allowed for each module in order to prevent concurrent access to registers of the target platform. In addition to function calls by the application, each module comprises a function that checks safety aspects related to this module. This function is called at least once per SafeOS cycle by the middleware (but may also be called by the application). The middleware modules ensure that all parameters used for function calls are within the specified ranges. Otherwise the middleware request the SafeOS to enter the safe state.

### 3 SIL3 Graphical IDE

In the previous sections the main components of our SIL3 environment are presented. The main questions of this section are:

1. **Who** is going to use the proposed environment?
2. **How** will an end user use the proposed environment?

So, this section starts by describing the main target group of end users of the SIL3 system-on-chip. In the main part of this section, a detailed description of the implementation and features of the graphical IDE is presented. Finally, the interaction between end user and the SIL3 system-on-chip will be explained.

#### 3.1 Target Group of End Users

The SIL3 system-on-chip is intended to be used for any application that may be aimed at up to SIL3. So, everyone interested in SIL3 certification of their system architecture is in the target group of interest. In addition, the SIL3 system-on-chip shall be able to carry out all possible tasks specified by the end user. The most crucial point: The end user wants to bring a system on the market as soon as possible. Of course, there also are plenty of other things that are coming into play. But those three points are compelling. Appropriately, the abstract group of end users is users that are interested in:

1. Having a SIL3 certified system
2. Having a lot of different features
3. Having a fast system architecture development

Then, who will really use the SIL3 system-on-chip? Starting from top to bottom, let’s say that the end user wants to have a SIL3 certified application. If defined safety features and communication features supplied by the target platform are enough for running the application, the end user then can choose the SIL3 system-on-chip as the starting point for designing his SIL3 application. Ultimately, end users of the SIL3 system-on-chip are all engineers designing SIL3 end products within the frame of abilities of the SIL3 system-on-chip.

The SIL3 system-on-chip has the SIL3 middleware as interface: the SIL3 middleware is written in C++. According to object oriented embedded programming, the SIL3 middleware is nothing else than only a small component of a bigger system. It is the interface between the SIL3 system-on-chip and the outer world. Using the middleware, only engineers with knowledge of C++ would be able to use the SIL3 system-on-chip. This way, the target group of end users would shrink pretty much. Not only, that just those engineers with good C++ background could use it, also they would need a lot of time porting existing solutions. Moreover, they would need to take care of other not so obvious things, such as:

1. Cycle time of operating system
2. Error handler of hardware and software components of the SIL3 system-on-chip
3. Correct task handle inspection
4. The SIL3 certification of the application code itself

All these tasks take time and cost. How to avoid all those problems? Using the SIL3 graphical IDE, the end user does not need to care about mentioned things. Those tasks are carried out by the SIL3 graphical IDE and the SIL3 middleware together with the SafeOS.

The user does not need to know C++. Target users are engineers who are familiar with ladder diagrams and function block diagrams. This is more straightforward and easier than dealing with C++ language constructs. Besides, the target group becomes wider as more engineers may be more comfortable with previous mentioned languages than with C++.

3.2 The SIL3 Graphical IDE

In this subsection communication and connection between SIL3 middleware and SIL3 graphical IDE will be explained in detail.

Fig. 2 represents an interaction diagram between the end user’s PC and the SIL3 system-on-chip. The SIL3 middleware represents an important link in the complete chain. It was designed before knowing any details about the SIL3 graphical IDE. Thus, the SIL3 graphical IDE can be omitted or replaced with another IDE. This is a very important fact, because full support for the SIL3 system-on-chip can also be provided by third parties.

The SIL3 graphical IDE can be installed on a PC like any other software program. The communication between PC and the SIL3 system-on-chip is via LAN-cable. The communication is bidirectional. In addition to downloading machine code to the SIL3 system-on-chip, the SIL3 graphical IDE is used for performing online tests (OLT) of the complete SIL3 system. A more detailed explanation of OLTs will be given in the following subsection.

A function block represents a single functionality of the SIL3 system-on-chip. For example, AND gate or a PWM output. A list of function blocks is stored in libraries of the SIL3 graphical IDE. Using function blocks the user can design a system behavior according to his needs. After designing the system, the function block diagram is compiled in machine code and downloaded to the target platform.

3.2.1 PC Side

In section 2.3, the list of the SIL3 middleware modules is given and briefly explained. The end user knows only about logic/function blocks. These blocks are linked with C++ classes of the SIL3
middleware. So, the IODevice module which is represented by IODevice class has its methods and attributes. The IODevice module in the SIL3 graphical IDE is represented by two blocks: I/O READ and I/O WRITE. By specifying the desired port for I/O READ block, the end user will get a value of the specified port as result. The SIL3 graphical IDE will call the following method in background during project building:

```cpp
theSIL3Middleware::IODevice::ReadInput(in_Port, out_Value)
```

When all function blocks in a diagram are interpreted, the SIL3 graphical IDE will link all codes together with a precompiled library of the SIL3 middleware. Then, the machine code is ready to be uploaded onto the target platform. Along with machine code, the SIL3 graphical IDE calculates a CRC32 of code and adds the timestamp. Machine code, CRC32 and timestamp are wrapped together, and then sent to the target platform via network.

### 3.2.2 System-on-Chip Side

Messages sent from the SIL3 graphical IDE will be received by COM system of the SIL3 system-on-chip via network. The COM system is non-safe. It acts like a black channel between the SIL3 graphical IDE and the SAFE system. The COM system and SAFE system of the SIL3 system-on-chip are connected via dual port RAM. The SafeOS provides safe communication end point. It implements a safety protocol stack. So, the sent message really has a turbulent path: from PC to the COM system and then to the final destination, SAFE system, via dual port RAM. The SafeOS verifies the CRC32 calculation, stores the timestamp and gives green light to the SAFE OS that machine code can be stored in FLASH. Notice that normally flash and RAM would be overwritten every time when a message from the SIL3 graphical IDE is received. That means, not only machine code received in previous messages but also SafeOS and the SIL3 middleware would be overwritten. This is not a good solution, as each time the end user wants to download his safe application to the target platform, he would also need to download SafeOS and the SIL3 middleware. Hence, flash and RAM of SAFE system are divided in three parts, as shown in Fig. 3. One part is used by SafeOS, the second part is used for the SIL3 middleware and the last, third, part is used for machine code received from the SIL3 graphic IDE.

Another advantage of separating flash and RAM in three distinct parts is that SafeOS can be updated without interfering user’s safe application or the SIL3 middleware. The link points between various flash and RAM parts are defined and they are always at fixed addresses.

### 3.4 Online Tests

The communication between the SIL3 graphical IDE and the SIL3 system-on-chip is not one-sided. The user has the option to design a complete industrial control system with the SIL3 graphical IDE. The SIL3 middleware and the SafeOS, as explained, are continuously performing a very large number of different tests. The SIL3 conformance is only possible by performing those tests. The user has the possibility to regularly get information about tests completeness and success. Moreover, for getting SIL3 certification the user shall also perform tests for his SIL3 application.

#### 3.4.1 The Online Test Communication

One test is testing the communication channel between the SIL3 graphical IDE and the safety system-on-chip.

To make safe communication, it can be implemented that SIL3 IDE sends a message along with its CRC32 value to the SIL3 system-on-chip. The SIL3 system-on-chip verifies the CRC32 value and sends a response to the SIL3 graphical IDE.

### 3.3 Interaction between IDE and End User

Simple examples of function block diagrams will be given in this section.

![Fig. 4: Graphical IDE function blocks for I/O READ and I/O WRITE](image-url)
The SIL3 system-on-chip has safe digital inputs and outputs. The SIL3 graphical IDE has two blocks that manipulate them: I/O READ and I/O WRITE. The user can add those blocks to the function block diagram with drag-and-drop action. After building and downloading code to the target platform, the user can monitor changes on safe digital inputs and outputs as shown in Fig. 4.

The next example describes how the state of the desired digital output port can be changed. Fig. 5 shows a function block diagram for a simple application. The hardware consists of 3 LEDs and 3 hardware switches connected to the SIL3 system-on-chip. The application monitors states of switches and combines states of switches with software input.

As result the LEDs are turned on or off. In the function block diagram, the LEDs are represented by the 3 blocks on the right side labeled LED_0 to LED_2. The software switches are labeled var_0 to var_2 and they are clickable. The states of the hardware switches are represented by SW_0 to SW_2. In this example, the user can watch what happens to the hardware; a blue wire represents an inactive state, a red wire represents an active state. Furthermore, the user can change the value of the variable of the application running on the target-platform. He can do that online, i.e. in the IDE while the test is running. In the example above, setting var_0 to “false” would inhibit the propagation of SW_0 to LED_0.

4 Conclusion

In our previous works [3][4][5], implementations of the SIL3 system-on-chip, the SafeOS and the SIL3 middleware are described. This paper concludes with the part of the SIL3 environment. It describes the SIL3 graphic IDE developed by our industry partner. The graphic IDE will be also SIL3 certified. Our contribution is adding support for the IDE to the SIL3 middleware. The main advantage of the SIL3 graphic IDE is that it enormously decreases development time of SIL3 applications. Moreover, without it, the user must be comfortable with programming in C++.

The aimed group of the end users are engineers who are developing SIL3 applications using PLCs. As PLC engineers may want to develop applications utilizing functional block diagrams or ladder diagram, support for the proposed SIL3 graphic IDE has been implemented. The SIL3 graphic IDE implementation is depicted. The communication channel between the SIL3 graphic IDE and the SIL3 system-on-chip is described. Major issues are described, e.g. segmentation of RAM. In the end, two simple application examples are given. With drag-and-drop actions the user can fast and easily develop SIL3 applications.

References:

