Introduction

Embedded systems are omnipresent companions in our daily life and they keep on emerging into new application fields. Namely “IoT” and “Industry 4.0” have many benefits. Embedded systems make our life more comfortable and safe. They improve the efficiency of traditional solutions and enable new use cases. On the contrary, this high degree of dependency makes us susceptible for any kind of misbehavior of these systems. Thus, the demand to protect ourselves from undesired behavior of embedded systems becomes an essential requirement for the design of new applications. This system level requirement propagates straight down to the heart of the embedded system, i.e. to the MCU, ASSP or SOC together with its software controlling the majority of system functions. The term “undesired behavior” refers to functional safety as well as to cyber security, since functional safety can only be attested for a genuine component which has not been altered or tampered in an unforeseen way.

Functional Safety is the part of the overall safety that depends on a system or equipment operating correctly in response to its inputs. The objective of functional safety is freedom from unacceptable risk of physical injury or of damage to the health of people either directly or indirectly (through damage to property or to the environment). Functional Safety is the detection of a potentially dangerous condition resulting in the activation of a protective or corrective device or mechanism to prevent hazardous events arising or providing mitigation to reduce the consequence of the hazardous event. Such effort is mandatory from ethical point of view and yields also economic advantages. EU-OSHA estimates that the cost of accidents at work and occupational illness ranges for most countries from 2.6 to 3.8% of Gross National Product (GNP). About 4.9 million accidents result in more than 3 days’ absence from work every year. Additional expenses like violations fines, higher insurance premiums, workers compensation, etc. come on top.

The prominent, fundamental standard for Functional Safety of Electronic systems is IEC 61508. Functional Safety certification in compliance to IEC 61508 is an enabler for current trends IoT and Industry 4.0 and emphasizes the commitment to product quality.

“Computer security, also known as cybersecurity or IT security, is the protection of information systems from theft or damage to the hardware, the software, and to the information on them, as well as from disruption or misdirection of the services they provide.”

Nowadays the risk for cyber-attacks drastically increases due to the cross linking of electronic systems via standardized interconnects, which is the heart of IoT and Industry 4.0 applications. This cross linking opens new flanks for cyberattacks not existent in the past. Due to the wide spread of electronic systems our susceptibility to failing or misbehaving components dramatically increases. The behavior of a compromised system is undetermined, thus its Functional Safety cannot be guaranteed anymore. Although IEC 61508 itself does not contain specific security requirements, it requests the assessment of potential security issues: “If security threats have been identified, then a vulnerability analysis should be undertaken in order to specify security requirements.”

IEC 27000 defines fundamental security requirements for general purpose IT security. Derived from it IEC 62443 defines security requirements for IA applications. A big difference between IEC 27000 and IEC 62443 is the priority of the protection goals. In IT systems for data processing the confidentiality of data has the highest priority while for IA applications system availability comes first.

There is also a difference between functional safety and cyber security standards. Safety standards define requirements, HOW a solution should look like. Cyber-security standards define requirements, WHAT has to be considered, but they don’t define how. The “how” is up to the user of the standard. Cybersecurity is a key factor to maintain functional safety in IoT and Industry 4.0 applications.

Life cycle management is one of the key requirements for functional safety as well as for cyber-security, but the focal point differs. For the implementation of functional safety a
thorough risk and impact analysis of all known threats is done once during concept phase. In next step, mitigation measures for these threats are designed, verified and certified. During the life cycle of a design the threat scenario is usually static. Life cycle management concentrates on the maintenance of the safety functions.

In case of cyber security an initial security-threat analysis is insufficient, since skills and means of attackers are constantly improving. Therefore, life cycle management of security functions means continuous reconfirmation of the actual security threats and steady improvement of the security capabilities of an application.

Functional Safety
The developer of Functional Safety related applications has to strictly obey safety constraints and rules defined by the applicable safety standards. This is an additional burden generating extra efforts when developing his application. Looking for synergies to share the work could mitigate these extra efforts. In best case, the development of a safety solution becomes less expensive and smoother than traditionally developed non-safe applications before. At this point the MCU vendor enters the stage.

Typically, the major portion of an embedded application provides foundational services like hardware drivers, task scheduling, network stacks, file system drivers, diagnostic functions etc. Here the application developer could benefit from deliverables of his MCU vendor by three advantages:

First of all his workload is reduced and he can concentrate on new features representing the real customer value of his application. Second, the MCU vendor is able to design optimized hardware related software functions, i.e. low level drivers or diagnostic functions, with his detailed knowledge of the device in mind. Last but not least the pre-certified components used by many other designs before have a high quality and a complete set of documentation reducing the total risk for certification of the target application.

Another aspect are the silicon related quality figures like FIT rates and recommended use cases. Without such data certification of the safety application cannot succeed. The component has to be proven in use or been developed following a compliant flow. The respective documents have to be submitted for a safety assessment of the target safety solution.

Designers of safety related applications should make sure in advance that the required data for all selected components can be obtained to avoid costly redesigns at a very late project stage.

The MCU vendor should provide a Safety Manual containing the device specific information. Contents of the Safety Manual are

- Safety SW capabilities
- Safety HW analysis
- Individual FIT rates for all MCU elements
- FIT rates for permanent and transient failure modes
- HW diagnostic measures

The Safety Software User Guide comes along with the safety library. It contains

- In-depth Software descriptions
- Implementation and integration guidelines
- Configuration Examples Safety Certificates
- The safety certificates confess the assessment result of the pre-certified software components.

Another aspect are the silicon related quality figures like FIT rates and recommended use cases. Without such data certification of the safety application cannot succeed. The component has to be proven in use or been developed following a compliant flow. The respective documents have to be submitted for a safety assessment of the target safety solution.

Designers of safety related applications should make sure in advance that the required data for all selected components can be obtained to avoid costly redesigns at a very late project stage.

Figure 1 – Deliverables to support safety applications

The MCU vendor should provide a Safety Manual containing the device specific information. Contents of the Safety Manual are

- Safety SW capabilities
- Safety HW analysis
- Individual FIT rates for all MCU elements
- FIT rates for permanent and transient failure modes
- HW diagnostic measures

The Safety Software User Guide comes along with the safety library. It contains

- In-depth Software descriptions
- Implementation and integration guidelines
- Configuration Examples Safety Certificates
- The safety certificates confess the assessment result of the pre-certified software components.

Figure 2 – Determination of functional safety parameters

Diagnostic
Hardware diagnostic an essential building block to confirm the continuous availability of safety functions during the whole life time of the solution. For components of redundant safety systems, the typical architecture for industrial control applications, the diagnostic functions need to have a medium fault coverage, e.g. >90% for SIL3, as well as a low runtime to execute within the process safety time of the target
application. Consequently, diagnostic should be supported by hardware functions and their software portion should be optimized for the respective MCU device. The role of the MCU vendor is obvious. The typical hardware diagnostic support of today’s MCU devices suited for safety applications are:

- **RAM parity**
  Parity protects against RAM data corruption due to transient and latent faults. CRC is an alternative to improve the availability of the application function.

- **WDT & Hardware timer**
  Timer mitigate system deadlock and violations of the task execution schedule due to tasks exceeding their time budget. The root cause could be either a software bug or a hardware related issue. In case of a failure, the WDT is able to issue a complete system reset while standard timers just inform by normal interrupts.

- **Clock Monitor**
  The clock supply is one of a digital circuit’s Achilles. Abnormalities of the clock like too high or too low frequencies or clock spikes could render the function of the circuit undetermined. The clock monitor circuit detects such incidents and issues a system reset to protect from undesired safety critical behavior.

- **ADC diagnostics**
  Many MCU applications use the integrated ADC to obtain safety related information from its analog environment. For a high diagnostic coverage of the integrated analog circuits, dedicated hardware support is desirable.

- **General purpose CRC unit**
  A general purpose CRC unit speeds up CRC calculations on data transmitted via interfaces or stored in the NVM of the device to minimize the runtime overhead. This helps to fulfill the fault tolerant time requirement.

- **MPU/MMU and bus guards**
  In case of applications consisting of safe and non-safe software components running on the same MCU, safe separation between the safe and non-safe domains is mandatory. To detect unauthorized access to memory or peripheral functions, online supervision of all bus masters and/or bus slaves is indispensable. This requires dedicated hardware like MPU/MMU or bus guards detecting and mitigating access violations immediately.

A certified safety library complements the hardware diagnostic functions of the MCU. It provides functions to control the hardware diagnostic functions and closes gaps not covered by them. The minimum requirement are self-test functions for RAM, ROM and CPU. As for all safety related software components life cycle management is a strong requirement. Updates and bug fixes must be guaranteed over the full life time of the target application. The root elements for diagnostics are the CPU core and its associated memories. These elements are commonly used by all customer applications. As efficient diagnostic of the complex CPU subsystem requires a deep knowledge of its construction, diagnostic functions should be provided by the CPU vendor treating the CPU subsystem as a Safety Element out of Context (SEooC).

The use cases of the MCU peripheral functions differ significantly depending on the target application and its system concept. Typically diagnostic of peripheral functions is not limited to the on chip circuitry of the MCU but involves external wiring and other components mounted on the PCB. Since the diagnostic concept must match the holistic function they test, a fixed test approach for diagnostics of peripheral functions makes no sense, i.e. peripheral functions can hardly be considered as SEooC and the task to specify and implement suitable diagnostic functions is up to the application designer. However, the MCU should provide supportive functions as mentioned above either by dedicated test functions or by a rich peripheral subsystem enabling redundant implementation e.g. by using two instances of a hardware timer simultaneously for the same purpose.

The diagnostic functions should be simple to use and easy to adapt to the runtime requirements of the target solution. Figure 4 shows a solution for the CPU self-test. It is divided into multiple independent test packages checking distinct portions of the CPU. This allows the user to run the basic test in between user tasks fulfilling his process scheduling requirements. All basic tests are written in assembler language.
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

The user application controls these basic tests via a common diagnostic API. The parameters passed to this API determine the basic test to be executed. After test execution the API returns the test result and the expected signatures. The API is written in C language making it easy to integrate into user applications.

CPU test software development – Proven Fault coverage

To meet SIL3 requirements of redundant systems, the target diagnostic coverage of the CPU self-test is 90% or above. Diagnostic coverage of 90% means that the CPU self-test will detect 90% of all possible faults in the digital logic circuitry making up the CPU function. This brings up the crucial point: how can the developer of the CPU self-test built up his test strategy and how can he confirm the resulting test coverage without knowledge of the CPU netlist? The MCU vendor is not allowed to disclose the MCU netlist, since it usually contains 3rd party IPs covered by NDA. But even if the netlist would be available, it requires a lot of expert know how as well as dedicated tools like fault simulators to obtain meaningful results.

To finally meet the target of 90% fault coverage a closed loop development strategy shown in Figure 6 is necessary. The fault coverage of the CPU self-test code is assessed. If it does not achieve the target coverage, thorough analysis of the uncovered circuitry detects the gaps in the self-test code. Needless to say that such analysis requires deep understanding of the circuit and its function as well as skills to operate the analysis tools efficiently. Next step is the improvement of the self-test code to close the revealed gaps and restart the fault simulation to determine the fault coverage of the updated code. This procedure continues until the target is met.

However, digital circuits like the CPU usually contain functions only needed for production test of the silicon chip or for application software debugging. Such logic cannot be stimulated by a functional CPU self-test. On the other hand, issues in this logic doesn’t affect the normal CPU operation. Therefore, it is permitted to exclude faults in this logic from the fault list. Again, the decision about which logic can be excluded required deep design knowledge only available at the MCU design team.

Figure 5 illustrates the importance to verify the achieved test coverage. It shows the evolution of fault coverage achieved when developing a CPU core self-test for a Renesas RX600 MCU targeting 90% fault coverage satisfying SIL3 applications. The initial code was done by experienced experts. All CPU commands had been stimulated, all CPU registers and calculation units have been triggered. This is the point where the development of a CPU self-test usually finishes. But a first fault simulation on the MCU netlist revealed a significant deficiency of the coverage really achieved. Far away from the 90% target a coverage of just 70% could be attested.

To improve the test coverage a closed loop development strategy is necessary. The fault coverage of the CPU self-test code is assessed. If it does not achieve the target coverage, thorough analysis of the uncovered circuitry detects the gaps in the self-test code. Next step is the improvement of the self-test code to close the revealed gaps and restart the fault simulation to determine the fault coverage of the updated code. This procedure continues until the target is met.

However, digital circuits like the CPU usually contain functions only needed for production test of the silicon chip or for application software debugging. Such logic cannot be stimulated by a functional CPU self-test. On the other hand, issues in this logic don’t affect the normal CPU operation. Therefore, it is permitted to exclude faults in this logic from the fault list. Again, the decision about which logic can be excluded required deep design knowledge only available at the MCU design team.
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

Fault simulation is a well-established methodology to confirm the test coverage of a test stimulus. The fault simulator executes two simulations in parallel: one for the unchanged, “golden” netlist and one for the modified netlist mimicking a fault at a certain logic gate. Input stimulus for both simulations is the self-test software to be assessed. The fault simulator compares the response of both simulations. If the test-software behaves different on the faulty netlist, the fault simulator counts the injected fault as covered, if the behavior of the test-software is same for both netlists, the fault is counted as uncovered. The observation point is the program counter, which can be checked by diagnostic software on the real silicon.

Simulation of about 200,000 faults in the CPU logic requires about 5 months per simulation. Parallel processing (dispatching to more than 50 processes) reduces the total runtime to a few days. At the end the achieved fault coverage can be quantified with a high level of confidence.

Safe Software Platform
A Safe Software Platform as shown in Figure 8 provides all basic system functions serving as a rock solid foundation of the target customer application. Traditionally, the application designer spends a lot of efforts to set up the basic system functions without creating real value for his specific application. In case of safety related systems this effort dramatically increases due to the additional requirements of a safety design. Using a pre-certified software platform containing all commonly needed base functions from hardware driver level and RTOS up to the middleware components for network and file system support gives him the opportunity to keenly concentrate on innovative application functions adding more value to his solutions.

A Safe Software Platform should not just comprise a conglomeration of pieces of source code. Its components should seamlessly build upon each other without gap. Integration of non-safe software components into a safe environment should be supported by temporal and spatial process isolation.

Certified device drivers abstracting control of the MCU’s specific hardware components build the ground level. A safety certified RTOS is an out-of-the-box solution to organize task scheduling and memory partitioning. Middleware components like TCP/IP network stacks, file system support, audio decoding and graphics libraries deliver high quality implementations of commonly demanded software functions. Since certification of safety certified components would be lost, if the user applies changes to them, he is not able to adapt them to his needs. Another disadvantage is the longer update cycle due to the safety assessment, which could be undesirable especially in case security flaws need to be fixed quickly. If not used for safety purposes, these middleware components can be integrated as sandboxed non-safe code. Such technology enables quick and straightforward changes to the sandboxed code without implications to the safety of the whole system.

Same as for middleware, the user application can either run as safety certified code or as sandboxed non-safe code. This allows to tailor the development efforts to the system requirements. Just the safety related functions have to be developed safety compliant. For legacy code or code supporting non-safe functions, e.g. a graphical user’s interface, sandboxing could be applied.

A suitable tool environment should be available along with the safe software platform. At least it consists of a safety certified compiler which is complemented by an IDE and potentially a code generator. The latter ones are not absolutely necessary, but they improve consistency to increase code quality by design and support structural separations of the software modules like memory partitioning and sandboxing.

As for all safety components, life cycle management is a must. The user should make sure to get software updates, bug reports and bug fixes during the whole lifetime of his own target project.
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

Figure 8 – Safe Software Platform
Cyber Security

Depending on the application field security of embedded systems has many flavors. The protection target as well as the necessary protection level depends on the specific use case. Availability, Integrity and Confidentiality are the focused security challenges of embedded applications. The security concept must be tailored for each application individually. Therefore, the protection demand for each security challenge has to be defined specifically for each use case.

A common, central requirement of secure systems is a secure, unique, trustworthy identity. Such identity is the root of trust enabling association of security related attributes to a specific device. Without secure identity common threats like address spoofing, man-in-the-middle attacks, counterfeiting etc. become trivial.

Attack skills and security levels

The skills, resources and motivations of hackers differ significantly. When planning security measures it is advisable to define against which type of hacker we have to defend our products, since this determines the efforts we have to spend. Typically attackers fall into one of the following groups:

- **Script kiddie**: driven by curiosity, for glory or the fun of it, low skills and resources
- **Professional**: targeting profit, medium to high skills, resources depend on potential swag
- **Governmental**: superordinate goal, high skills, almost unlimited resources

The security standards define different levels of security to correspond to the application’s security requirements and the attacker’s motivation. For example [4] defines four security levels:

- **SL 1** – Prevent the unauthorized disclosure of information via eavesdropping or casual exposure.
- **SL 2** – Prevent the unauthorized disclosure of information to an entity actively searching for it using simple means with low resources, generic skills and low motivation.
- **SL 3** – Prevent the unauthorized disclosure of information to an entity actively searching for it using sophisticated means with moderate resources, IACS specific skills and moderate motivation.
- **SL 4** – Prevent the unauthorized disclosure of information to an entity actively searching for it using sophisticated means with extended resources, IACS specific skills and high motivation.

Depending on the required security level, adequate security functions should be implemented in hardware and software. If the application has a clever security concept, protection demand of its components can be reduced. For example, if hacking of a single device doesn’t reveal common encryption keys enabling a class break, basic tamper protection of the MCU hardware might be sufficient. This kind of gradual security feature selection paves the way to a cost optimized system solution. A step up in the Security Level of a system significantly increases the efforts for design and maintenance. It complicates debugging and verification introducing additional schedule risks. So the adequate choice of the targeted Security Level is crucial for the success of a new product.

Unfortunately, the skills of the three groups of attackers are continuously improving. Hacker tools exploiting known vulnerabilities become more and more sophisticated and can be operated even by low skilled attackers. This makes life cycle management indispensable. The security related functions of an embedded solution should be continuously improved to close known gaps. A strategy to minimize the efforts per application is efficient reuse of software, e.g. by a software platform concept as already discussed in the safety context, see Figure 8. With the adoption of a widely used software platform, a single user benefits from the experience of a large community leading to a high maturity level of the system functions. Bugs and security flaws are fixed just once by the provider of the software platform and become available to each single user without the need for the respective resources on his side.
Developing a secure application – Threats and Countermeasures

Security starts at the system concept. Following the defense in depth approach, a single protection measure is insufficient. Depending on the protection requirement, multiple defense lines should be prepared under the assumption that the outer shield has been broken.

It is crucial to document the basic requirements and solution concepts as detailed as necessary to avoid ambiguities and misunderstandings and enable in-depth review of security functions. The detailed user’s manual of the MCU with comprehensive descriptions of all hardware functions provides necessary information to define a secure system concept. Any missing or ambiguous description creates a risk for overlooking possible attack scenarios.

The major risks in the implementation phase are flaws and backdoors introduced by designers, intentionally or unintentionally. To minimize the risk, only trusted and security conscious resources should be engaged. Access rights to the source code repositories must be strictly controlled. Especially write access has to be restricted to guard against code manipulation. The security related program code should be thoroughly reviewed. Where ever available matured and trusted library functions should be used rather than self-written code to improve quality and robustness of the software. In best case a matured software platform ranging from driver level up to middleware functions is used. Support of crypto experts is advisable, especially for the implementation of crypto functions. Typically such experts are involved for the design of platform functions.

For application debugging and profiling, the developer needs full transparency and visibility of the internals of the MCU. Sophisticated run control, memory contents manipulation and trace functions are basic features of today’s devices. But these salutary mechanisms could become an easy breach of the system security opening the door for reverse engineering and tampering via standard development tools. Therefore, a fundamental security feature provided by the MCU hardware should be secure disable of debug interfaces. The lock-out of any development tool should be irreversible as long as the application code and data is present in the nonvolatile memory of the MCU. Depending on the use case of the application it could be beneficial to lock such access continuously to securely prevent operation of the application unit with non-authorized firmware. The firmware of a locked MCU could still be maintained by a secure firmware update mechanism preventing the download of manipulated code or eavesdropping.

Code injection, data corruption or eavesdropping are serious attack scenarios listed according to their severity. Injected code could permanently manipulate the affected device and serve as an entry point to the system, which could attack other components from the inside, e.g. over a network link. Corrupted data could temporarily or permanently change the system behavior. Eavesdropping could reveal secret information, which is already a threat by itself. Furthermore the obtained data could be used to perform more extensive attacks subsequently.

There are several options to defeat such attacks. First of all each system component should possess a trusted identity confirmed by cryptographic functions. This trusted identity should be protected against modification and cloning. The silicon device should have a secure nonvolatile memory region, which can store the necessary secret information in a temper proof way. The program code performing security related functions should be signed to detect manipulations. At least the boot process should verify this signature and lock access to security related system functions, if the signature is invalid. A secure firmware update mechanism enables firmware updates in unprotected environments and over public communication links while maintaining system security.

Sensitive information exchanged via physically accessible interfaces should be at least signed by cryptographic functions to detect manipulations during the transmission. Secret information must be encrypted. A trusted identity of the communication partners is mandatory to enable a secure exchange process of the data encryption keys between them. Without securing the right counterpart for key exchange a man in the middle attack could easily breach the data security efforts.

Secure and Unique Device ID – The root of trust

As already emphasized in this text, a secure and unique identity of devices is the solid fundament to the successful implementation of security measures in an embedded system. Today, most embedded applications are built using standard devices, which are produced and sold in high volumes. The high volume production has many benefits like high product quality, an extensive ecosystem, good availability and a competitive pricing. On the other hand, counterfeiters have the same unlimited access to these standard off the shelf components. This enables them to manufacture identical clones of the genuine equipment, which behave exactly same as the genuine ones. Even if the OEM programs his devices with an ID, the counterfeiter can do same. In the field it is hard to differentiate, if the respective ID is a valid one or if it is copied or faked.
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

As a countermeasure of this threat, the silicon device is initialized with his own, unique ID right at the time of its birth in the silicon fab making it identifiable throughout its whole life. With such individual ID, which cannot be replaced, even off-the-shelf products are distinguishable from each other. Still a counterfeiter could obtain off the shelf components for his clones, but he is no more able to copy the valid ID of a genuine product.

If the ID of the device is stored in a centralized server of the silicon vendor, the background of all devices becomes transparent and their history can be traced consistently and continuously from their birth down to their disposal. The registered, unique ID serves as a secure root of trust and enables advanced services on top of it. Examples are secure firmware provisioning and updating, trusted and secure communication and secure life cycle management.

Enabling Integrity - Securing Memory

The embedded system’s memory contains the application code, which determines the behavior of the controlled device, as well as all volatile and non-volatile data the system acquires, processes and stores. This is why attackers usually have a high interest to find out about the memory contents and gain the ability to manipulate them according to their needs. Our goal must be to keep full control over the memory contents and access to them during the full life cycle of the device. Especially when operating the MCU we must ensure the memory protection from reset release until power off as depicted in Figure 11.

It is the responsibility of the first stage boot loader, which is the first piece of software executed after CPU reset release, to perform all sanity checks and prove that the memory contents are valid and unchanged. This inhibits the execution of invalid firmware images and makes malware injection more challenging. The first stage boot loader itself is stored inside the MCU in a ROM type memory and cannot be modified.

Next the first stage boot loader enables all access protection mechanisms especially to the secret information hidden in the device. Then the user level device management and RTOS functions are starting. They run in a privileged mode and have the ability to perform system control and management tasks, e.g. controlling the memory access permissions of user application code. These functions should have a high maturity level and their quality shall be thoroughly verified and reviewed.

Last but not least the user application code runs with lowest permissions under control of the RTOS. It can only access to memory regions as granted by the RTOS and is not permitted to redefine the properties of these regions. This avoids interference with other user application level tasks and fends any attempt to impair security related system properties.

Despite the essential boot and permission control there are still multiple ways for an attacker to gain unauthorized memory access. The most obvious gateway into the device would be the debug interface used by the application developer to test and improve his software. If not securely disabled or physically protected, the attacker has a plug-and-play access to the MCU just by using a standard development tool environment. The next level are security flaws or backdoors in the software. Although they are more difficult to find, they can be easily exploited once they are known.

But even if the debug interfaces are closed and no (known) software vulnerability is present, there are still physical attacks like the so called “side channel” attacks left. The
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

ingredients to build a secure system protected against physical attacks have a big overlap with the ingredients for a functional safe system. In particular, all measures improving system robustness are also effective countermeasures against software exploits and side channel attacks to the hardware. In a typical offense scenario, the attacker tries to deploy such methods to manipulate the program flow without the need of changing the well protected program memory contents. Manipulating the program flow could for example skip password checks. The probability for a successful attack can be minimized by stringent supervision of the program flow and its behavior supported by the following system components:

- **Memory Protection Units (MPU)**
  They firewall areas of memory prohibiting access by non-privileged code and mitigate the risk of fraudulent or faulted code accessing sensitive data.

- **Dedicated hardware isolating and protecting sensitive memory regions and ensuring immutable secure boot.**

- **Hardware write protection to system control registers.**
  Enforcing strict separation between application functions and device management.

- **Dedicated Bus Slave Protection circuits or adoption of the ARM TrustZone® concept to support security domains only accessible by trusted code.**

- **Temporal supervision of the program execution timing by a Window WDT.**

- **Accelerator circuits for CRC or hash calculation to periodically confirm checksums or cryptographic hashes across sensitive memory regions.**

The listed items also accelerate the software development process and help to increase product quality, since they immediately reveal software bugs like pointer errors, which are otherwise hard to debug.

**Hardware Security Features**

Hardware security features fall into two classes: security functions like crypto-accelerators and tamper protection like secure key storage. As a response to the increasing security demand of today’s and future applications, hardware support for security becomes a standard MCU function. Table 1 outlines a cost conscious selection of crypto functions determined by application targets as found on standard MCU devices today.

Besides securing the JTAG debug access a symmetric encryption accelerator, a true random number generator and a unique device ID are the minimum security assets to build up a root of trust inside the MCU device. Already this minimum feature set enables secure boot, secure identification and secure data exchange including secure firmware updates. Performance and feature set are limited. Missing functions could be emulated by software. However the limited system resources of low-cost devices and the low tamper protection of such solutions render them less useful.

Low end devices add the HASH accelerator hardware function. It enables authentication of exchanged data with higher throughput.

For Mid- and High End devices asymmetric cryptography and a secure storage for user defined keys complement the basic functions. These features enable additional and more flexible use cases like secure communication with arbitrary communication partners on the network.

**Product Lifecycle Management and Secure Manufacturing**

– foundation to maintain the integrity

Secure life cycle management of firmware and crypto keys is a basic precondition to establish and maintain security throughout the whole product life time. The security requirements in the development and manufacturing phases of the target product are obvious, especially in cases where the product is assembled by a contract manufacturer outside of the OEM’s premises. In such cases the OEM usually leaves all production relevant information to his third party. In such case it is a challenge to keep control about the production flow and quantities, i.e. the relationship between OEM and his contract manufacturer is based on overreliance.

Figure 12 shows a solution for secure manufacturing under such conditions. The OEM encrypts his software for the target appliance before releasing it to the manufacturing sites. The MCU vendor injects the secret software decryption key into the devices ordered by the OEM. Upon programming the genuine MCU ordered by the respective OEM can decrypt the OEM’s software and securely store it in its on-chip memory.

<table>
<thead>
<tr>
<th>Function</th>
<th>High End</th>
<th>Mid End</th>
<th>Low End</th>
<th>Low Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symmetric Cryptograph Accelerator</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Asymmetric Cryptograph Accelerator</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Hash Accelerator</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>True Random Number Generator</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Asymmetric Key Generation Accelerator</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Key Secure Storage</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Limited JTAG Access</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Unique ID</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

Table 1 – Hardware security features by performance class
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

Such concept has multiple advantages: it avoids reverse engineering of the application code, impedes tampering of the code, avoids overproduction and allows an easy check for authenticity of the product in the field.

Managing Safety & Security
Safety and Security have an intrinsic dependency: there is no safety without security! Safety is attested to well-designed and documented technical implementations. Each and every change of the safety system enforces a diligent assessment of the changes and a recertification of the system. Security issues open the door for corruption of the safety system through a cyber-attack rendering the safety certification inapplicable.

IEC 61508

IEC 61508-1
1.2 In particular, this standard
...k) requires malevolent and unauthorised actions to be considered during hazard and risk analysis. The scope of the analysis includes all relevant safety lifecycle phases; NOTE 5 Other IEC/ISO standards address this subject in depth; see ISO/IEC/TR 19791 and IEC 62443 series.

However, in the past safety related control systems usually operated in isolated islands separated from other office equipment and networks. Cyber-attacks focused on PCs and data centers. So there was no need to further complicate development of safety equipment by adding cyber-security requirements on top of the multitude of safety requirements. Today, this picture changed drastically. There is the vast trend to enable remote monitoring and control of each and every unit including those in safety related domains. So today cyber-security is an inevitable requirement for safety equipment. Unfortunately, the system designer faces contradicting requirements for achieving functional safety and cyber-security. A manageable solution solves this dilemma by separating functional safety and security functions from each other’s. This could be achieved on PCB level by implementing both functions in separated devices like safety

Figure 12 – Secure Manufacturing
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

applications usually do when providing two independent safety channels. On the first glance such solutions looks bullet proof and rather simple. But it has drawbacks: the component count increases, the interface between security- and safety functions opens a new flank for reverse engineering making a mutual check of safety- and security functions less reliable.

Integrating functional safety- and cyber-security into one silicon device while maintaining their independence is a more preferable solution.

**MCU Architecture - Separation of safe, Secure and other functions**

Today’s IA components typically integrate three major functional domains:

1. Rich user functions, e.g. sophisticated process control algorithms, a complex GUI, OPC-UA server and extensive network communication
2. Functional safety related features protecting humans and the environment
3. Security functions safeguarding the integrity and availability of the system and protecting sensitive data

Table 2 lists the diverging requirements the software of these domains has to fulfill. It is desirable to split development of the corresponding software modules to reduce complexity and concentrate the design efforts on their key requirements.

<table>
<thead>
<tr>
<th>Code Size</th>
<th>User Functions</th>
<th>Security Functions</th>
<th>Safety Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>Very complex</td>
<td>As simple as possible</td>
<td>As simple as possible</td>
</tr>
<tr>
<td>Development flow</td>
<td>No restriction</td>
<td>Security compliant</td>
<td>Safety compliant</td>
</tr>
<tr>
<td>Update Cycle</td>
<td>Medium, on demand of customers or new market requirements</td>
<td>Fast response to new security hazards</td>
<td>Slow / never, typically the risk scenario is static</td>
</tr>
<tr>
<td>Hazard if compromised</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Hazard on malfunction</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Verification Level</td>
<td>Medium</td>
<td>Medium to high</td>
<td>Very high</td>
</tr>
<tr>
<td>Certification</td>
<td>None</td>
<td>None (as of today)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2 – Properties of software domains

A pure software based approach could not assure safe and secure separation of the three domains and requires all software components to have the same level of safety and security. Hardware support is necessary to achieve and assure independence of them and allow their independent treatment.

Safety (IEC 61508-3)

- As far as practicable the design shall minimize the safety-related part of the software.
- Where the software is to implement both safety and non-safety functions, then all of the software shall be treated as safety-related, unless adequate independence between the functions can be demonstrated in the design.
- Where the software is to implement safety functions of different safety integrity levels, then all of the software shall be treated as belonging to the highest safety integrity level, unless adequate independence between the safety functions of the different safety integrity levels can be shown in the design. The justification for independence shall be documented.

Security

- Restrict access to confidential information to the security related program code (least privileged)
- Protect security related program code from being tampered (integrity)

The most efficient approach to manage the divergent requirements of rich user functions, of up-to-date security functions and of certified safety functions is a hardware platform, which supports stringent separation of these three domains from each other’s. This principle is called “sandboxing”. Each of the introduced code domains gets its own sandbox, a protected environment with well-defined interfaces for information exchange, in which it acts undisturbed from other code and without interfering to other functions. Sandboxing refers to hardware supported isolation of processes, isolation of code and isolation of data. This is very similar to threads supported by an OS except the stringent hardware supervision.

Who controls this hardware supervision gets the next crucial question. The security related code cannot tolerate supervision by non-secure code, because this would allow to bypass security functions. The safety related code in turn cannot tolerate supervision by not safety certified code, since this non-safe code could interfere with the safety function. So there is only one way out: for the
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems

separation and control of the three functional domains we need code, which is safe and secure. This is not a contradiction to the discussed separation requirement, since this code – typically called Hypervisor - doesn’t implement any security nor safety function by itself. Instead, it just takes care about a safe and secure isolation between them, i.e. it manages the independence of the code domains. So the Hypervisor can be compact and of low complexity enabling safety certification. Due to its low complexity and functionality the surface for cyber-attacks is pretty limited.

Figure 13 – Safe and secure isolation

Figure 13 shows the principal outline of a system controlled by a Hypervisor providing three sandboxes, one for each discussed software domain. In this architecture all three code domains run in their own environments. This concept safely and securely prevents any side effects on the safety code as well as on the security code. They are both isolated from each other and from the rest of the application code, which may be neither safe nor secure. The advantages of this concept is, that application code and security related code can be updated at any time without endangering the safety functions. Application code can neither break the safety function nor can it circumvent the security functions. The hypervisor strictly controls all communication between the sandboxes.

There are several hardware solutions supporting sandboxing, which supplement or reinforce each other’s:
- System with MMU/MPU on all bus masters
- Bus Guards protecting sensitive targets
- Flash access guard protecting reading and programming of sensitive Flash regions
- ARM TrustZone®
- Multi-core devices with strict logical boundaries between the core domains

There are as many different solution approaches as application use cases exist. So the distinct technical solution may vary from system to system depending on the concrete requirements.

Literature
[2] IEC 61508-1
[3] IEC 62443-1-1
[4] IEC 62443-3-3
Comprehensive Solutions from MCU Vendors to support Developers in Design of Safe & Secure Embedded Systems