



Testing Embedded Systems


Quality Assurance in Embedded Software Development


Identify and reduce effort through better forecasts, risk minimization and demand-based capacity planning

| | |
|-------|-------------------|
| 02 | Introduction |
| 03-08 | The process |
| 09 | Predictability |
| 10-11 | Risk minimization |
| 12-15 | Needs assessment |
| 16-19 | Risk management |
| 20-25 | Business Case |
| 27 | Literature review |

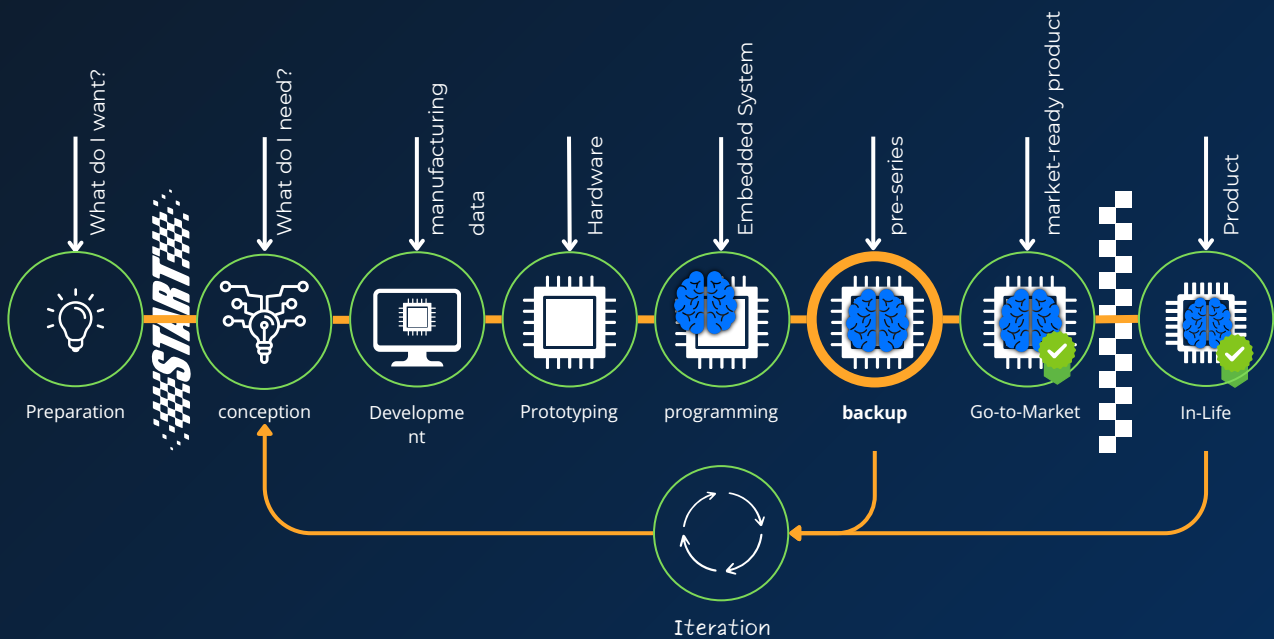
 [+49 \(0\) 176 814 561 07](tel:+49(0)17681456107)

 kontakt@essaar.de

 +49 (0) 176 814 561 07

 essaar.de/kontakt

Introduction



Quality assurance of development

Identify, investigate and optimize aspects of effort

Quality assurance is essential in electronics design. However, it is often associated with high and unpredictable costs. The experiences and insights described in this article are based on observations over several years and have helped us to identify, analyse and ultimately reduce the effort required for functional and performance testing*. As a result, we have been able to optimise our development costs and time, improve time-to-market and ensure higher product quality.

To share these results, we first provide a basic and generalised description of the quality assurance process in electronics development. The aspects of predictability, risk minimisation and demand-driven capacity planning are then examined individually. By suggesting a qualitative analysis of the needs and capacities of your own quality assurance, you can estimate whether and to what extent the effort can be optimised. The key question is how well the process requirements are covered by the available capacities. Finally, a business case is used to show which specific optimisations can be achieved. The es:scope® software is presented in the context of the document. The appendix contains an overview of the literature on testing embedded systems.

From a bird's eye view

1. Setup



2. The test



3. Matching



4. Adjust



Quality assurance of development

Identify, investigate and reduce hidden costs

During development, requirements are defined, implemented and tested - the VDE V model is the underlying standard. In an ideal world, the implementation would directly satisfy all defined requirements, making testing unnecessary. But even with precisely formulated requirements and carefully performed calculations and simulations, quality assurance often reveals deviations from the desired behaviour. This is because calculations and simulations always make idealised assumptions about reality. Factors such as material differences, environmental influences or unforeseen interactions in the system mean that real-world testing is required. Why is that? Because the complexity of real systems exceeds the accuracy of models. Measurements are therefore essential - to adapt models to reality, to check the functionality of the developed systems and to correct deviations from the desired behaviour.

During quality assurance, it may happen that, despite adjustments, the desired results are not achieved. In such cases, a design iteration may be required to correct the deficiencies. At the same time, the validation process may identify certain functions as redundant. These functions can be simplified or even eliminated in a simple iteration to make the system more cost effective.

Terminology

Terms such as "testing" and "adaptation" are often too vague to accurately describe the specific processes in the development workflow. To avoid misunderstandings, the key concepts that play a central role in development are explained: verification, validation, calibration and matching.

Verification:

Verification refers to the quantitative testing of whether specified requirements are met. It is an objective measurement that must be maintained within a specified accuracy - for example, a measurement deviation must not exceed 5%. The aim of verification is to ensure that the developed system operates within the defined tolerance limits.

Validation:

Validation goes beyond verification and looks at whether the right product has been developed to provide the desired solution. It focuses on whether the features developed meet the real need - for example, whether five USB ports on a laptop are really necessary or whether the feature is over-dimensioned. Validation ensures that the features created are useful and necessary in the real application.

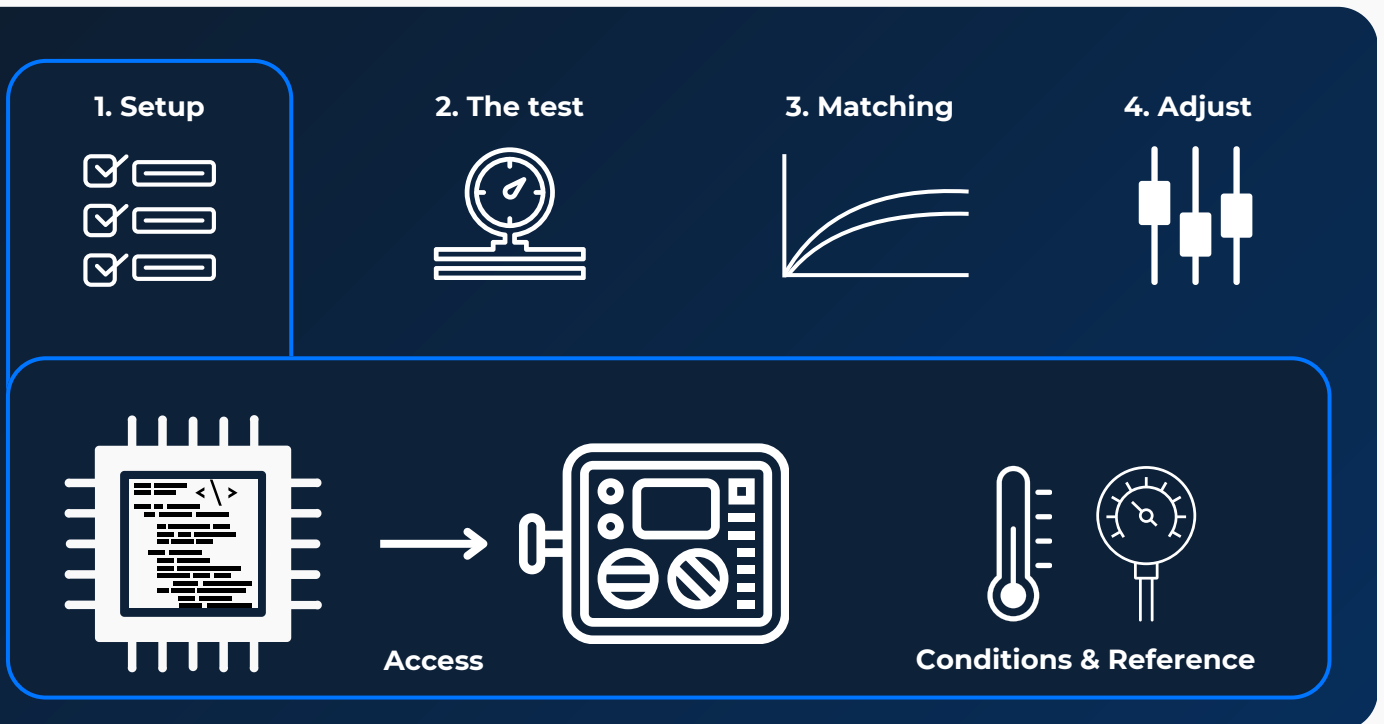
Calibration:

Calibration is the adjustment of system parameters to correct for systematic deviations. A typical example is the pre-processing of a sensor signal to compensate for offset and improve measurement accuracy. These adjustments are usually made using software by resetting calibration values or parameters.

Tuning:

Tuning goes beyond calibration and refers to the fine tuning of parameters to optimise system performance. A typical example is the smooth starting behaviour of an engine achieved by precise adjustment of the control parameters. Tuning requires successful calibration and aims to optimise the performance of the system under the given conditions.

The quality assurance process 1/4



1. Setup

measurement access, test conditions, reference measurement

To perform tests, both the physical system and a measurement system must be available and correctly set up. This presents several challenges:

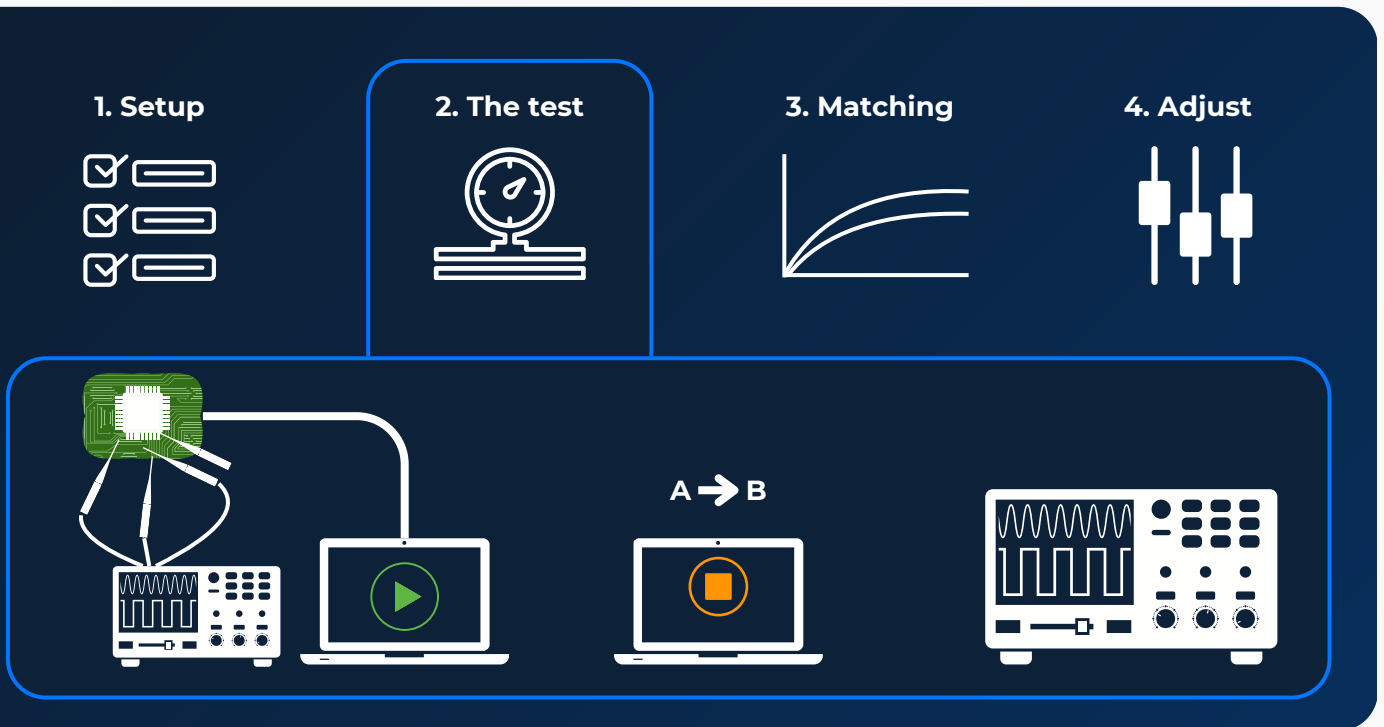
Access: The quantities to be measured must be made accessible to the measurement system. Internal signals from a microcontroller, for example, need to be brought out, which can be difficult with limited computing power and a limited number of outputs.

Test conditions: Special conditions such as extreme temperatures (e.g. testing in an oven) can further complicate testing.

Reference measurement: A reliable reference measurement is required to compare the test results to a value.

System resources: Limited storage space and communication channels can make data collection difficult. If data is sent via communication protocols such as CAN or UART, special solutions for data transmission often need to be developed.

The quality assurance process 2/4



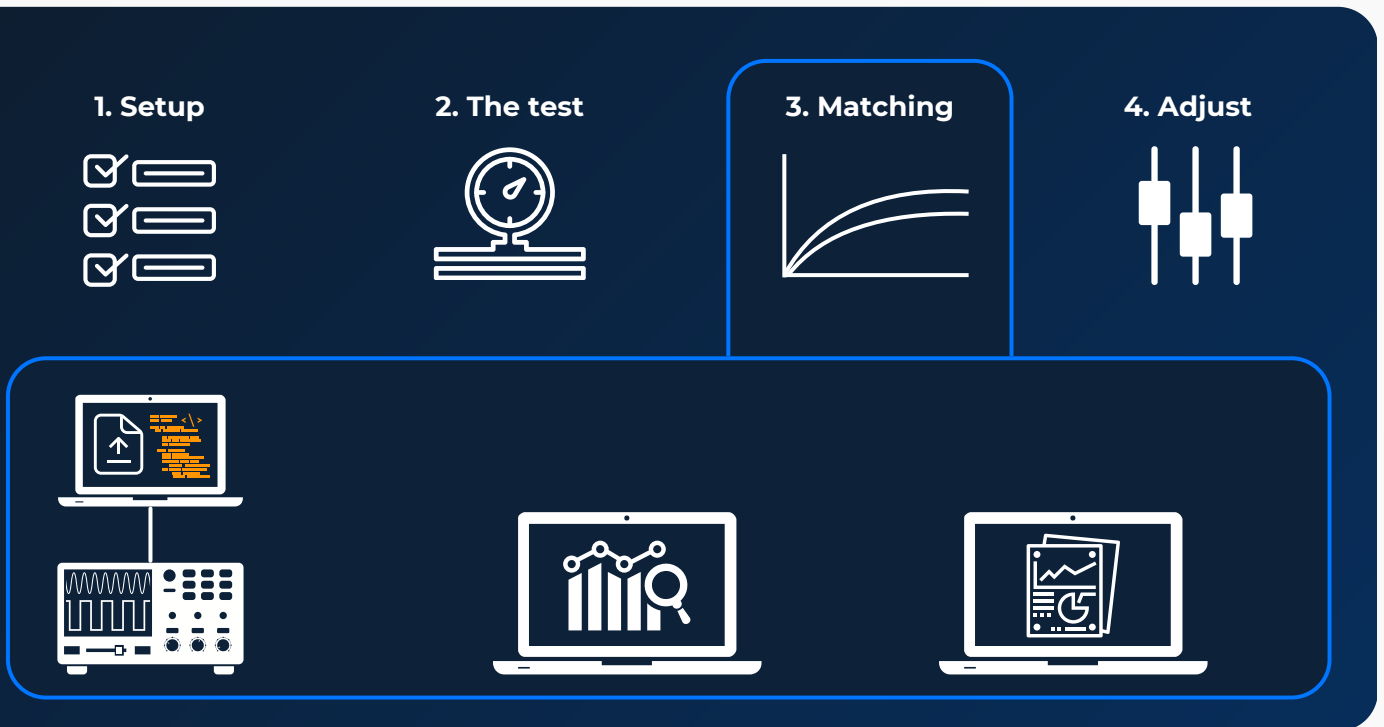
2. The test

start of operation, state change, monitoring

Once set up, testing begins, which can take anywhere from a few minutes to several days, depending on the complexity and requirements. There are several challenges to overcome during this step:

- Start up: The system is started under the defined test conditions and the accessible signals are monitored or recorded with the test system.
- Data acquisition: The relevant measurements are recorded.
- Synchronisation: Unsynchronised data will lead to misinterpretation.
- State change interrupts: Tests can be interrupted by disturbances or deliberate changes in operating conditions, requiring flexible monitoring.
- Data rate: High data rates can make transmission and storage difficult, especially when system resources are limited.
- Data quality: The usability of measurement results depends on the accuracy of the measurement system and the synchronicity of the data.
- Test monitoring: Misconfigurations are often discovered after the fact, as real-time monitoring is not always possible.

The quality assurance process 3/4



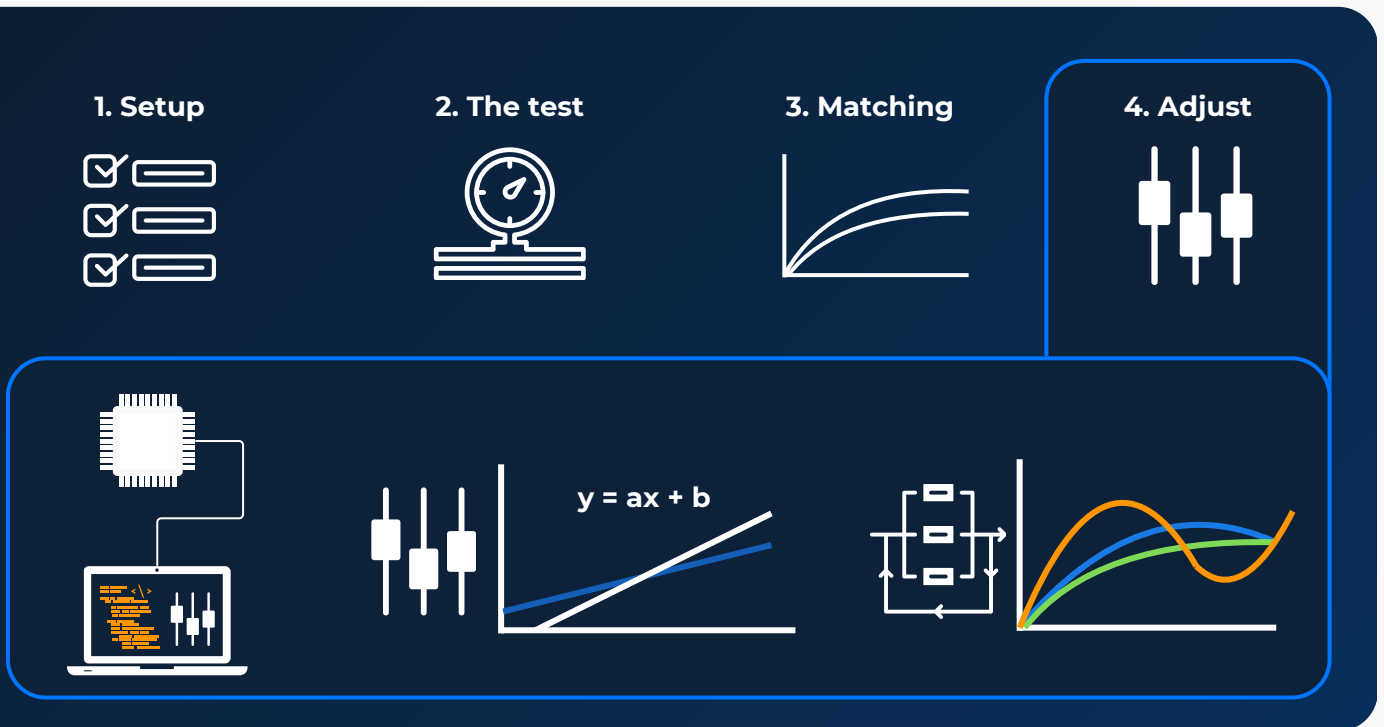
3. Match:

Export, Analyze, Evaluate

After the test, the data collected is analysed. This step includes

- Data export and analysis: The test data is exported and processed in analysis software (e.g. MATLAB).
- Verification: The system behaviour is compared with the expected values and the reference measurements to ensure that the requirements are met.
- Validation: The measurement data is used to evaluate whether the developed system meets the requirements and fulfils the desired functions.
- Failure analysis: If deviations occur, it is investigated whether they are caused by measurement errors, model inaccuracies or system-related problems.
- Comparison and evaluation: Deviations are identified by comparing them with existing models and reference data, which provides insight into necessary adjustments to system parameters.

The Quality Assurance Process 4/4



4. Adjust:

Adjust, Calibrate, Tune

Based on the results of the analysis, the system parameters are calibrated and tuned to correct deviations and optimise the system. This step includes

Software adjustments and hardware changes: Depending on the nature of the deviations, both software adjustments (e.g. resetting parameters) and hardware modifications (e.g. replacing components) may be required.

Calibration: System parameters are readjusted to correct systematic deviations and ensure accurate operation.

Tuning: After calibration, parameters are fine-tuned to maximise system performance under real operating conditions.

Iterative cycle: The whole process is repeated until the system meets all the validation requirements. The number of iterations and interruptions can vary greatly depending on the complexity of the system.

Predictability of the effort

Our experience in embedded systems development has shown that considering all the factors that influence quality assurance is critical to optimising project planning. If all cost and time aspects are not taken into account during project planning, the actual effort is almost always underestimated.

Transparency in the project plan: It is important that quality assurance is listed as a separate and detailed item in the project plan. Often the testing process is seen as part of the general development tasks, which leads to the actual effort being underestimated. A simple solution to this is to create separate schedules and budgets for specific quality assurance activities (e.g. calibration, prototype testing, software verification). This will ensure that each test process is realistically accounted for and reduce the risk of budget overruns.

Consider the iterative nature of the process: An important lesson to be learned when planning for quality assurance is that its iterative nature makes it difficult to predict, even as a stand-alone item in the project plan. Buffer times and flexible test cycles have proved useful in responding to unforeseen problems. Process interruptions and changing responsibilities can delay the planned project duration. The actual number of iterations of adaptation loops is also difficult to predict. The following business case shows how we have reduced the number of interruptions and iterations with es:scope®.

Data quality tailored to your needs: The reliability of decisions depends heavily on the quality of the data available. If measurements are inaccurate, asynchronous, incomplete or fail to capture all relevant information, important deviations may be missed or misinterpreted. Such problems can in turn lead to quality defects in the final product. Investing in high quality measurement and test systems tailored to your needs will pay dividends in the long run.

Weigh the risks of penalties for poor quality: Deficiencies in quality assurance that lead to defects in serial production pose significant financial and reputational risks. Unlike software, which can be updated, improvements to physical products are much more complex. In the long term, the benefits of thorough quality assurance outweigh the short-term savings. It is important to set clear priorities and strike an informed balance between speed and quality to avoid problems down the road.

Risk minimization through TDD

In application software development, test-driven development (TDD) is a proven method of creating tests before the actual code is written. Test coverage can then ensure that no problems occur after the code has been released. In the development of embedded systems, TDD is possible in principle, but is much more complex. The reasons for this are

Hardware dependency: Embedded system development is closely linked to the underlying hardware. While emulators such as QEMU make it possible to simulate hardware environments, certain hardware specifics such as electrical signals or physical interactions are difficult or impossible to emulate. Critical aspects such as signal transmission accuracy and timing play an important role in real-time systems and require testing on real hardware. This reliance on physical hardware limits the applicability of TDD, especially when it comes to testing the interaction of software and hardware in different environments.

Customised development processes and environments: Embedded systems are often very specific and diverse, making it difficult to standardise TDD. Unlike application software, where there are established toolchains, tools for embedded systems are often less mature or difficult to access. Different hardware platforms, customer-specific requirements and different toolchains make it difficult to implement a standardised TDD process. The diversity of development environments leads to increased adaptation efforts to integrate TDD into the process.

Resource constraints: Embedded systems are often constrained by limited memory, processing power and I/O capabilities. This presents a challenge as many TDD tests rely on extensive resources to validate code coverage and functionality.

Real-time requirements: Many embedded systems operate in real-time, which means they must meet strict timing requirements. Testing real-time requirements is particularly challenging because test environments often cannot replicate this behaviour with the required accuracy. Even minimal delays in simulated environments can produce results that differ significantly from reality, compromising the validity of the tests.

Black, Grey and White box testing

Black-box, grey-box and white-box testing methods differ greatly in how deeply testers can penetrate into the internal details of the software. Embedded systems present additional challenges due to the close integration of hardware and software - there is software that depends on the hardware.

Black box testing focuses on the external interfaces and visible functionality of the system. The tester has no insight into the internal processes or source code. This is particularly useful for dynamic testing, which tests the system's response to various inputs at runtime (including fuzzing). For example, you can test the behaviour of a sensor by exposing it to different environmental conditions and measuring the system's response. The advantage is that the tester is testing the system like an end user, but insight into internal bugs or time-critical issues is limited.

White box testing, on the other hand, provides a detailed view of the source code and hardware implementation, and enables both static and dynamic testing. Static tests examine the code without executing it to identify errors in the logic or possible vulnerabilities. Static analysis tools scan the source code for security holes or unfulfilled conditions. Dynamic tests in white box testing focus on run-time aspects, such as checking interrupt handlers or register accesses in the microcontroller. Emulation also plays an important role here, allowing tests to be run before the real hardware is available. QEMU (Quick Emulator) is often used to simulate the hardware environment to test the behaviour of the software, especially in the early stages of development. However, emulation has its limitations when it comes to hardware-critical aspects such as signal accuracy or real-time behaviour, so final testing on real hardware is required.

Grey box testing combines the approaches of black box and white box testing and is particularly useful in scenarios where the tester has partial knowledge of the source code and hardware. Tests can be performed to specifically test critical modules or interfaces. In embedded systems, grey box testing is often used to test specific components or communication protocols such as UART or SPI, where both external functionality and internal logic are relevant. Dynamic testing can be used to ensure that the system functions correctly under different operating conditions.

The combination of dynamic and static testing within the three test methods is essential in embedded systems, as the tight coupling of software and hardware requires special test approaches. While black-box tests ensure that the system functionally meets the requirements, white-box and grey-box tests help to detect deeper-level bugs that may arise due to the close hardware dependency and real-time requirements.

Demand & capacity



Determine the need for quality assurance

Examine aspects of the need

The need for quality assurance depends heavily on product requirements and business strategy. Long-term optimisation of quality assurance is worthwhile if it contributes to sustainable improvements in product quality and operational efficiency. The decision as to which quality assurance measures are appropriate depends directly on the type of product, the application environment and the long-term goals of the company. The determination of requirements is based on a cost-benefit analysis that carefully weighs the cost and benefits of testing activities to create a quality assurance capability that meets the requirements.

A company that rarely develops new products could invest in basic but cost-effective quality assurance, while high-throughput companies with customised solutions require more extensive, automated testing and more dynamic quality assurance measures to cope with the large number of variants and requirements. For less complex systems, such as a temperature sensor, a minimalist test strategy aimed at simple functional testing is sufficient, while complex systems, such as the power management in an electric excavator, require robust quality assurance that covers different scenarios, load changes and extreme conditions.

Need for quality assurance

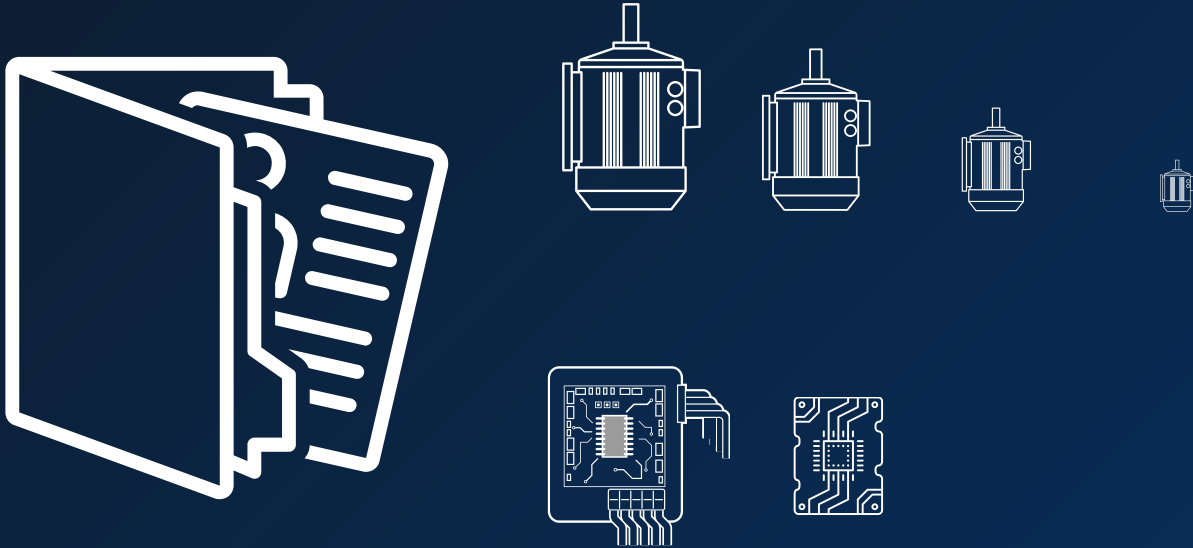


1. Project-related requirements

Technical requirements of the product

- Number of requirements: The more requirements that need to be covered, the greater the effort required to record and evaluate these variables. The correlation between the parameters and their different levels of complexity has a significant impact on the test effort.
- Number of devices and test cycles: The number of devices to be tested affects the effort required to perform the tests - especially when test configurations have to be set up on a test bench. Multiple quality assurance iterations, which are often required, significantly increase the effort, especially if each test device has to be reconfigured.
- Interface diversity: The large number of communication interfaces (e.g. CAN, SPI, UART) can lead to increased test effort, as each interface must be specifically tested.
- Software complexity: The more complex the firmware, the greater the impact on effort, as complex programs often have more edge cases to test.
- Data rate: Data rate requirements can have a major impact on the scope of quality assurance. For example, if very short system response times or current peaks at 10 kHz need to be measured, this will require special measurement systems and test procedures.
- Real-time requirements: If the product must meet certain real-time requirements, the test systems must also be designed to accurately test these requirements.
- Resource limitations: High data rates or large amounts of test data may be hampered by limited communication channels, storage space or computing power. Specific measures must be taken to overcome these limitations.
- Operating modes and environments: The number of operating modes and special operating conditions (e.g. extreme temperatures) have a significant impact on the quality assurance effort. Tests under real operating conditions are often more complex and require special equipment.

Need for quality assurance



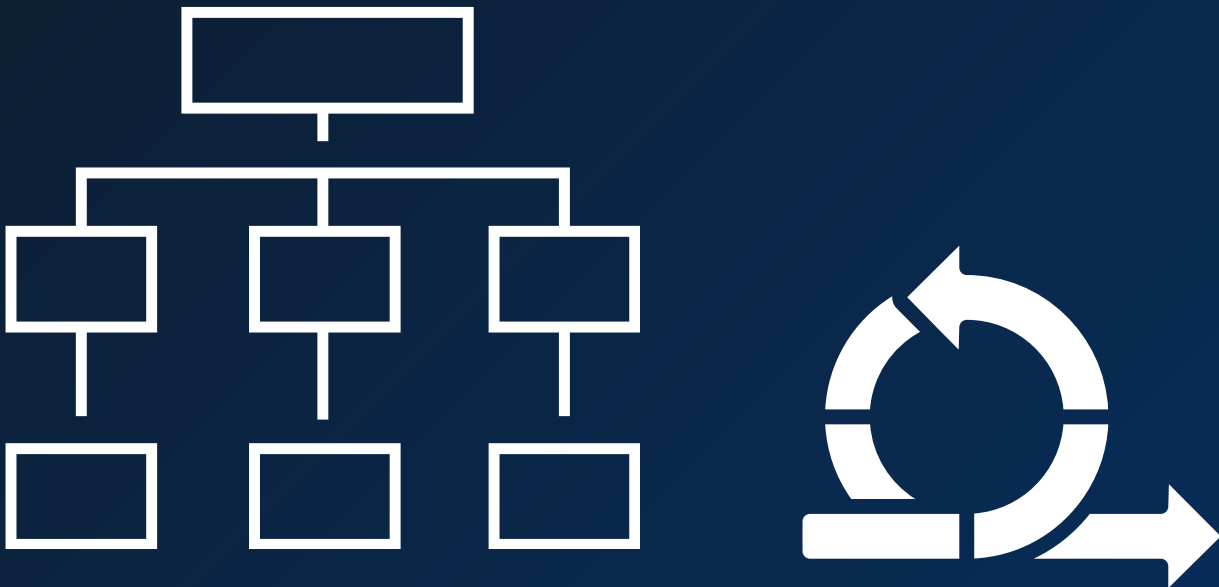
2. Portfolio-related requirements

product management and business requirements

A company's product portfolio also has a major impact on the need for quality assurance:

- Highly specialised products: If the portfolio contains only a single product with very high technical requirements, it may be acceptable for quality assurance to be inefficient because the frequency of testing is low.
- Product vs. project business: Companies in the product business tend to perform quality assurance runs less frequently than companies in the project business, which are constantly developing new, customised solutions.

Need for quality assurance



3. Strategic and operational requirements

corporate structure and processes

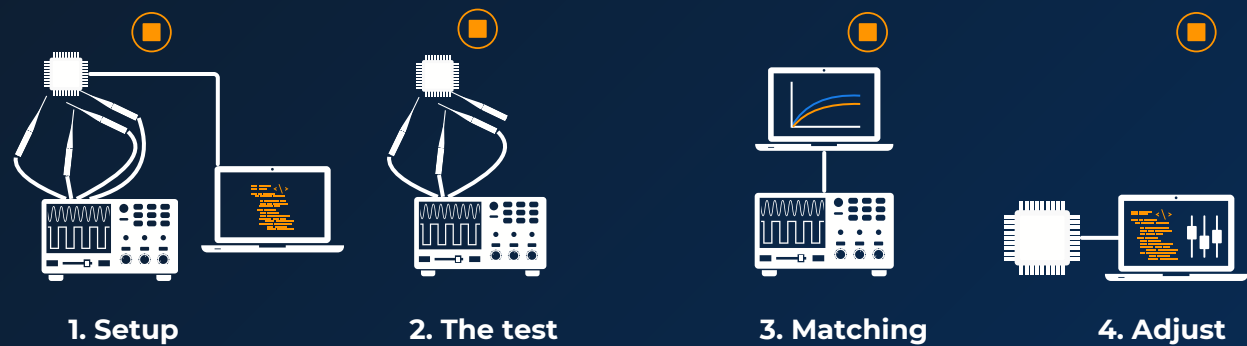
The organisational structure also influences how effective quality assurance can be. A cross-departmental process landscape can be particularly challenging.

When the development of a product is spread across multiple departments, such as hardware, software and validation, iteration loops can be particularly problematic. For example, a developer in the validation department may discover an anomaly and send the product back to the programmer. This often leads to additional delays, especially if these departments have different priorities or parallel projects. In such a scenario, quality assurance is a process that needs to be both well structured and agile to be efficient.

Capacity assessment

Quality assurance from handicrafts to space travel

The efficiency of the tuning process depends on the processes, the tools available and the number of staff. Every company has different capacities. Here we have five arbitrarily defined levels to classify the capacity of quality assurance.



Level 1 (Basic): Black Box, Offline Adaptation, Asynchronous Capture

Description: Laboratory equipment, such as an oscilloscope, is used to acquire data from the pins of the microcontroller or at test points.

Features: Data is exported and analysed after a test. Adjustments are programmed manually. The data from the laboratory instrument is not synchronised with the inputs or other system parameters. This approach is very time consuming and iterative, as adjustments are only made after the system is switched off and the data is evaluated manually. Each adjustment requires a new test run, resulting in long development cycles. The black box approach with asynchronous data allows corrections to be made for major deviations from the measurement data calibration, but is of little help in tuning controllers.

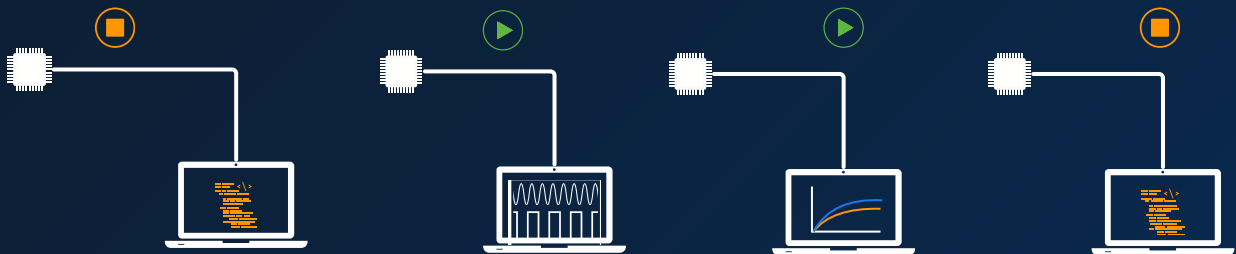
Application scenarios: Suitable for simple systems with very low requirements where only occasional adjustments are required.

Example: A temperature sensor that serially outputs a rounded temperature value every 10 minutes.



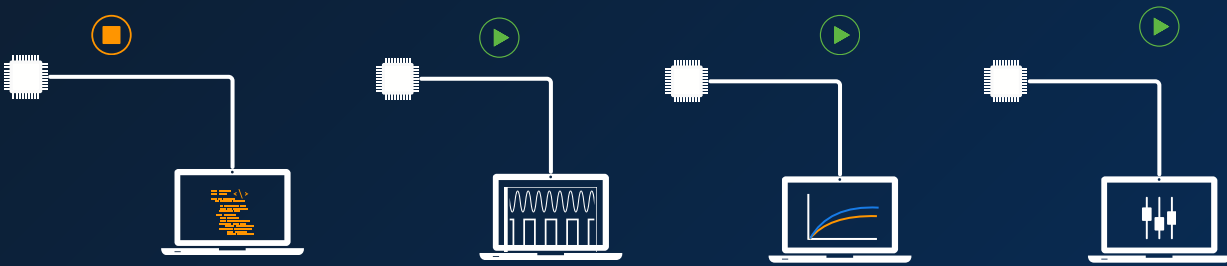
Level 2 (time synchronisation): Black Box, Offline Adjustment, Synchronised Acquisition

- Description: Data acquisition is performed using a synchronised data logger to which internal data is output via a communication interface.
- Features: After the test run, the data is exported and analysed to determine the necessary adjustments. The sampling rate is high enough to monitor overall system performance, but details may be lost during rapid system changes. Collecting data with a data logger improves efficiency compared to Level 1, but the offline nature of tuning remains, resulting in many iterations. Simple tuning tasks are possible.
- Application scenarios: Simple sensor applications
- Example: Test and evaluate the creep behaviour of a capacitive pressure sensor under different operating conditions for inclusion in a look-up table.



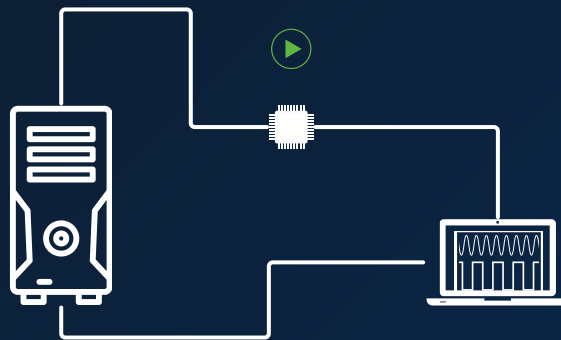
Level 3 (real time): Grey Box, Offline Adjustment, Synchronised Real-Time Recording

- Description: Acquisition and recording of selected variables is performed while they are displayed and monitored in real time and during ongoing system operation.
- Features: Conclusions about system behaviour can be drawn directly during the test and parameter adjustments can then be programmed into the software without separate analysis. Multiple data sets can be directly compared. There are fewer uncertainties and interruptions than with recording without real-time display.
- Application scenarios: Tuning of controllers, validation with a large number of signals, high test coverage through unit tests, systems with high requirements and high quality standards.
- Example: FOC motor control of a BLDC controller



Level 4 (online actions): Grey box, online customisation, synchronised real-time capture

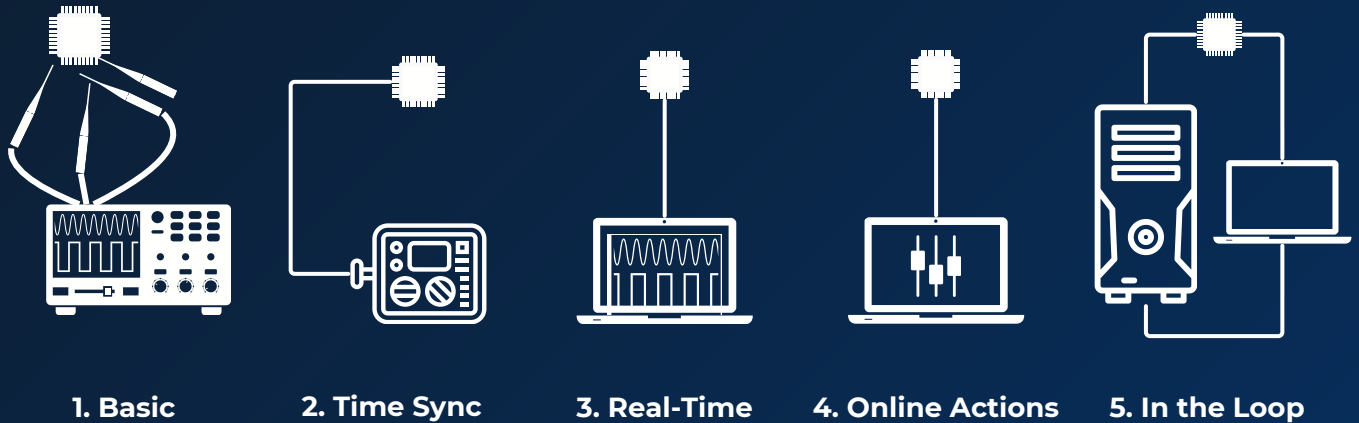
- Description: Same conditions as level 3, but with the ability to set parameters, send commands and set event triggers at run time. These triggers activate data collection when specific events or anomalies occur, allowing critical system conditions to be captured.
- Features: Runtime adjustments reduce the number of iterations and interruptions. The high sampling rate enables very accurate monitoring of the system. Event triggers can be used to capture rare or critical events in the system, making the tuning process more efficient and targeted.
- Use cases: Particularly useful for systems that require high test coverage, that need to operate under different environmental conditions, or whose performance needs to be highly optimised.
- Example: Sensorless FOC motor control of a BLDC controller



Level 5 (In the Loop): White box testing with hardware-in-the-loop (HIL)

- Description: At this level, the tuning process is fully automated. A HIL (hardware-in-the-loop) system allows parameters to be automatically adjusted in real time and immediate optimisations to be made.
- Features: Automatic tuning and continuous real-time monitoring allow complex scenarios to be tested, with the system responding to simulated environments in real time. The set-up effort is high, both in terms of cost and technical implementation.
- Application scenarios: Ideal for safety-critical applications (e.g. aerospace, medical) where failure could have serious consequences.
- Example: An autonomous vehicle control system can be tested to simulate how the vehicle reacts in different situations (rain, snow).

Capacity assessment



Five levels of capacity

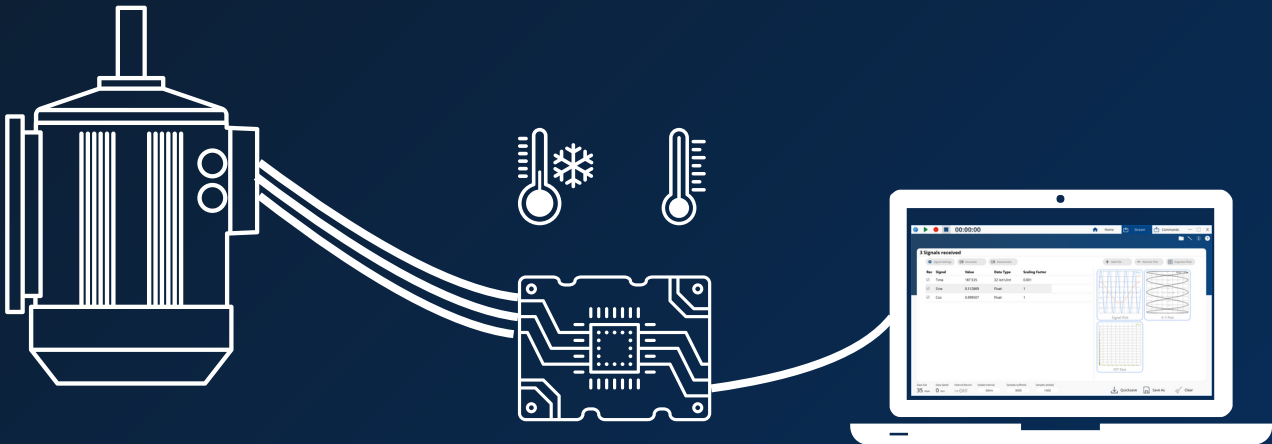
Quality assurance from handicrafts to space travel

Level 1 or 2: Sufficient for simple, non-critical systems, but with the uncertainty that the backup effort is difficult to estimate and can be large in the worst case. The tuning process is relatively inefficient here, so the effort can grow quickly if many iterations are required.

Level 3 or 4: Essential for complex, demanding systems in time-critical projects. The use of real-time data and synchronised adjustments ensures more efficient process control and better predictability of the assurance effort. This can significantly improve development time and quality assurance.

Level 5: Due to the high cost and effort involved, this is only appropriate for security-critical and highly complex applications. Such systems often need to operate under a wide range of operating conditions and environmental factors, requiring extensive automatic real-time monitoring and adaptation.

The Business Case: es:scope®



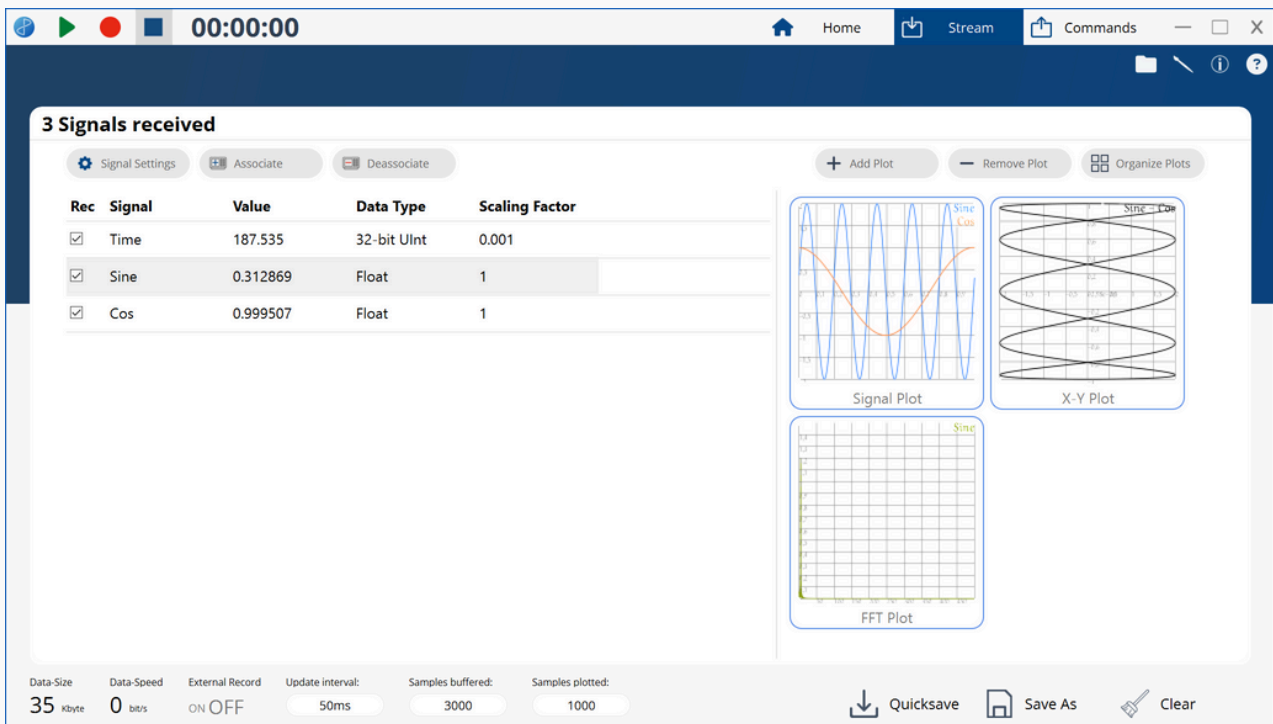
The initial situation

Developing new engine control in medium-sized companies

A medium-sized company is developing a new FOC motor controller for a BLDC motor with strict real-time requirements and limited system resources. The current development process is considered efficient enough, but still time consuming due to multiple iterations and manual adjustments. The data used for validation is considered to be sufficient, although test coverage is not high.

- **Technical requirements:**
 - FOC motor controller for BLDC motor
 - 20 parameters in the requirement list
 - High data rate (20 kHz for control, 1 Mbps in the CAN bus)
 - Minimal system resources
 - Environmental conditions -20°C to 60°C
- **product portfolio:**
 - Drive systems such as inverters, motors, motor controls and Hall sensors, with a typical service life of 8-12 years
- **Corporate structure:**
 - Medium-sized company with 160 employees
 - Separation between software and hardware development; validation by electrical engineers
- **Current quality assurance capacity:**
 - Using a data logger to collect data
 - After each iteration, new parameters must be programmed and tested manually
 - Simulations support parameter selection, but the system must be shut down regularly

The Business Case: es:scope®



The approach

Introduce and test es:scope® in the existing development process

The aim is to integrate es:scope® into the existing development process in order to increase efficiency and to evaluate the benefits for subsequent projects. A software engineer and a hardware engineer are licensed to determine the best way to implement and use the software.

Process:

- After a short introduction, es:prot is integrated into the existing microcontroller code.
- Relevant variables for the measured quantities and parameters for calibration, as well as the communication parameters are selected.
- Real-time data can now be tracked on the laptop and parameters adjusted via any serial interface, even while the device is in the temperature test chamber.
- Once a valid configuration is found, the code is finally adjusted to complete the development.

Benefits:

For the electrical engineer: Real-time analysis of the measurement signals and controller performance, as well as calibration and tuning during the test.

For the programmer: Unit testing of hardware-dependent software that was previously not possible.

1. Time expenditure & labor costs

By using es:scope®, iterations can be reduced and lead times can be significantly shortened. Previous manual adjustments and system shutdowns are no longer necessary as es:scope® enables real-time analysis and calibration.

- Real-time analysis and calibration: Adjustments are made while the system is running, eliminating the need for system shutdowns.
- Efficient parameter selection: Measurement parameters can be set by simple code changes, minimising set-up and adjustment times.
- Universal interfaces: es:scope® can communicate with various protocols, eliminating the need for complex adaptations and speeding up setup.
- Plug and Play: Pre-configurations for the display of signals make commissioning easier.
- No reboots required: Operating modes can be changed directly by commands from es:scope®.
- Number of tested devices and test runs: The test configuration described in the software allows a large number of devices to be analysed via plug-and-play.

Example:

| Aspect | Previously | Afterward |
|---------------------------------|---------------------|---------------------|
| Setting up the measuring system | 4 hours | 1 hour |
| Wage costs of the setup | 320€ | 80€ |
| Iterationen | 8 iterations | 2 iterations |
| Time required per iteration | 8 hours | 2 hours |
| Total labor costs per iteration | 640 € (8h * 80 €/h) | 160 € (2h * 80 €/h) |
| Total costs | 5440€ | 400€ |

Better planning: By reducing iterations and speeding up setup, there are no interruptions caused by other tasks, enabling accurate project planning.

Process acceleration: 2 weeks

Savings in labour costs: 5080€

2. Equipment costs

The result: Procurement costs can be reduced with es:scope

By using es:scope®, the procurement and maintenance requirements for specialised measurement technology can be significantly reduced. In many cases, expensive systems such as hardware-in-the-loop (HIL) systems, specialised data loggers or other test equipment become superfluous. Existing hardware and software is used to achieve the required test accuracy and data quality.

Reducing the need for measurement equipment es:scope® enables optimal use of existing infrastructure, minimising the need for expensive new investments in measurement technology:

- Less hardware: Instead of data loggers, digital oscilloscopes or HIL systems, real-time data acquisition is enabled directly on a computer, reducing the dependency on required physical measurement devices.
- Less software: With the es:prot protocol working with es:scope®, there is no need to develop additional microcontroller code for signal conditioning, measurement protocols or special software for data formatting. This not only saves development time, but also avoids potential sources of error when creating your own software solutions.
- Fewer peripherals: es:scope® allows the existing system to be used for data acquisition without the need for additional external measurement technology or communication peripherals. Since es:prot can be integrated directly into the embedded software, it offers great flexibility in system architecture and eliminates the need for additional physical interfaces or devices.
- Less maintenance: Unlike physical devices, es:scope® eliminates hardware maintenance costs.

Some examples:

To illustrate the savings potential through the use of es:scope®, here is an example calculation:

Cost of a data logger: €1,000 to €5,000.

Development costs for own microcontroller software for signal provision: 40 hours * 80 €/h = 3.200 €.

Total savings:

The use of es:scope® reduces the acquisition costs for measuring equipment, as es:scope® offers a cost-effective alternative to conventional measuring technology. A key feature is its interface independence, which reduces the complexity of adapting the system and test benches for specific tests. Material costs that would normally be incurred for such adaptations are eliminated or minimised. Development costs for your own test software can also be reduced as es:scope® provides a comprehensive, ready-to-use test and analysis environment.

3. Quality risk

Result: The higher test reliability and test coverage with es:scope® can increase the certainty that design errors are detected before production. The programmable triggers and high sampling rate of es:scope® make it possible to identify and document errors, anomalies and failure scenarios.

- Detection of anomalies, defects and failures: With the high sampling rate of es:scope®, even the smallest deviations from the desired system behaviour can be detected. Critical conditions and peaks that might go undetected at lower sampling rates are detected early. es:scope® also allows for long-term testing where critical system conditions are recorded. Programmable triggers can be used to track anomalies and document the exact conditions under which they occur. Commands can be used to test the response of the system.
- Fine tuning and optimisation: Real-time feedback of measured variables and on-line parameter adjustments allow different settings to be quickly tried on the real device. This makes it possible to find a parameter configuration for optimised system performance.
- Test coverage: With es:scope®, hardware-dependent software can be analysed in different test scenarios. Commands can be used to check the system behaviour for specific inputs. Flexible parameter selection, command-based changes to system behaviour and real-time analysis reduce the likelihood of discovering bugs or errors after production.

Example:

To illustrate the savings potential, here are examples of typical cost points:

- Cost of rework: Typical rework costs can run into five figures, especially in volume production. These costs include not only materials and labour, but often additional testing and, if necessary, the return of defective units.
- Costs for faulty hardware: If a repair is not technically possible, the full material and production costs are incurred without compensation. This is especially true for complex embedded hardware, where failures are often irreversible.
- Potential downtime costs: Production downtime can easily cost tens of thousands of dollars per day, depending on the volume and urgency of production.
- Production delays: Delays due to unexpected failures can delay time-to-market, resulting in lost sales.

Sources that address the economic impact of test coverage:

- Jovanović, Z., & Živković, D. (2006). "Testing of Embedded Software". *Journal of Automatic Control*, 16(1), 9–14.
- National Institute of Standards and Technology (NIST) (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing*.
- Zhu, H., Hall, P.A.V., & May, J.H.R. (1997). "Software Unit Test Coverage and Adequacy". *ACM Computing Surveys*, 29(4), 366–427.
- Vahid Garousi et al. *Testing embedded software: A survey of the literature*, 2018

Conclusion

1. Setup



2. The test



3. Matching



4. Adjust



Identify & reduce effort

Better forecasting, risk minimization and demand-based capacity planning

This document provides a comprehensive overview of the quality assurance process in development. In order to reduce time and investment, predictability, risk minimisation and demand-oriented capacity planning of quality assurance were examined in detail.

With es:scope®, validation and calibration processes can be optimised, costs reduced and time-to-market shortened. The use of es:scope® usually pays for itself from the very first project. In the long term, the savings increase with each subsequent project, so that the benefits grow continuously. Both embedded software engineers and electrical engineers benefit from the ability to optimise their workflows and improve test coverage.

The aspects of quality assurance effort discussed in this document can be specifically addressed by using es:scope®. However, each individual case should be examined to determine the specific requirements for quality assurance and the extent to which an expansion of capacity makes sense. We would be happy to assist you in creating a tailor-made business case to evaluate the optimal use of es:scope® for your company.

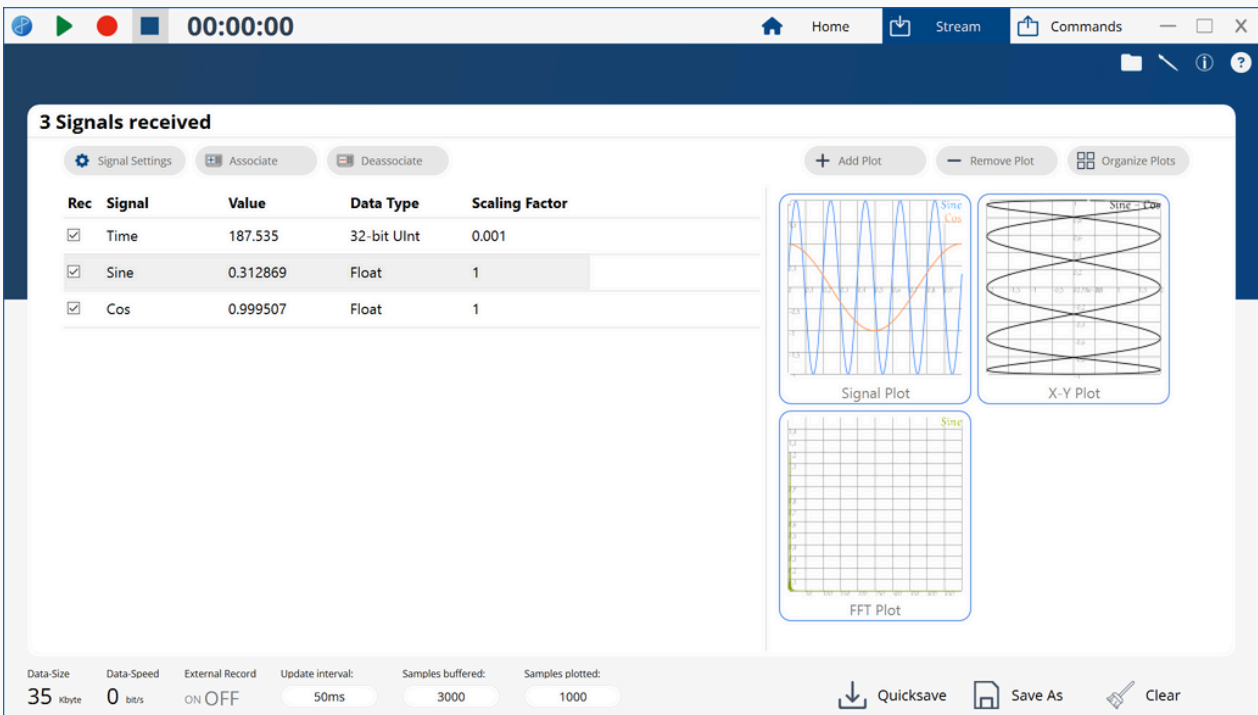
Contact

 <https://wa.me/4917681456107>

 kontakt@essaar.de

 [+49 \(0\) 176 814 561 07](tel:+49(0)17681456107)

 essaar.de/kontakt



Request es:scope® trial version now

essaar.de/esscope

Appendix A: Testing Embedded Software

The publication "Testing Embedded Software: A Survey of the Literature" by Garousi et al. (2018 [https://doi.org/10.1016/j.infsof.2018.06.016]) provides a comprehensive overview of the state of the art in embedded software testing. Several important findings on the state of quality assurance and the challenges in testing embedded systems can be derived from this study:

1. Specific challenges when testing embedded software:

- Hardware-software integration: Embedded systems are characterized by the close integration of hardware and software, which makes testing more complex.
- Resource constraints: Limited memory, computing power and energy influence the choice of testing methods.
- Real-time requirements: Time-critical functions require special testing approaches to identify timing and synchronization problems.
- Safety-critical applications: In domains such as automotive, medicine or aviation, high reliability requirements must be met.

2. Variety of test methods:

- Model-based testing: Using models to generate test cases and simulate system behavior.
- Hardware-in-the-Loop (HIL) Testing: Testing with real hardware in a controlled environment to check the interaction between hardware and software.
- Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) Testing: Simulation of hardware components to perform software tests independently of the physical hardware.
- Formal verification: Application of mathematical methods to prove properties of the software.
- Static and dynamic analysis: Analysis of the source code without (static) and with (dynamic) execution to detect errors at an early stage.

3. Automation and tool support:

- Test automation: Necessary to increase efficiency and reduce human errors. Automated testing enables more frequent and thorough testing.
- Specialized tools: Development and use of tools tailored to the specific requirements of embedded systems.

4. Test coverage and quality metrics:

- Increased test coverage: Higher coverage leads to better fault detection, but is often difficult to achieve due to the complexity of embedded systems.
- Metrics: Using metrics to evaluate test effectiveness and identify high-risk areas.

5. Challenges and open research questions:

- Scalability: Dealing with the increasing complexity and size of embedded software.
- Automated test case generation: Need for methods to automatically generate test cases, especially for rare or critical scenarios.
- Integrating tests into the development process: promoting continuous integration and continuous testing practices.

Conclusions on the state of the art:

- Need for integrated approaches: Effective quality assurance in embedded systems requires integrated testing strategies that consider hardware and software together.
- Early testing: Shift-left principles, where testing begins early in the development cycle, are critical to detecting and fixing defects early.
- Combination of methods: A combination of different testing methods (static, dynamic, formal) offers the best results.
- Resource awareness: Tests must be designed taking into account the resource constraints of embedded systems.