WHITE PAPER

Best Practices for Targeting AUTOSAR Classic and ISO 26262 with Simulink

MathWorks®

The automotive industry is going through a massive change in relation to the complexity of the systems being developed and the speed at which they are deployed. Many of the features of these systems are for automated driving and active safety.

Specifically, the importance of software design has grown due to the exponential increase in the feature set functionality for production vehicles. This in turn increases the size, complexity, and overall difficulty in delivering the software that goes into a single vehicle. The deployment speed and time to market are all moving at an unprecedented pace. Most OEMs are now offering continuous over-the-air software upgrades on a regular cadence. To deal with this software and electrical system complexity, there is a need for a structured approach on how to manage and design these electrical systems. For this reason, many automotive OEMs and suppliers have decided to standardize on ASPICE for QM components and ISO® 26262 for ASIL components. The remainder of this paper will focus on ISO 26262 compliance and best practices. ISO 26262 has 12 sections with 10 normative parts and two guidelines that help organizations create a consistent process for developing high-integrity software and electrical systems. The normative parts provide requirements that cover a variety of topics such as high-level analysis of safety risks, system and architecture design considerations, and implementation-level HW/SW design considerations.

AUTOSAR's intent is to create a structured approach to managing an electrical system's software architecture and construction. This approach inherently lends itself to a top-down design methodology that is conducive to ISO 26262 compliance for application software. This paper explores the best practices in AUTOSAR that can help with ISO 26262 compliance when using Model-Based Design.

How Can AUTOSAR Help with ISO 26262 Compliance?

AUTOSAR was developed with functional safety in mind. This is shown by the document released by AUTOSAR consortium titled *Overview of Functional Safety Measures in AUTOSAR*. This document describes crucial concepts and constructs that are applicable to safety-related applications. Many of these methodologies will be referenced throughout this paper. Examples include but are not limited to:

- Well-defined architectural constructs
- Software reuse
- Encapsulation of algorithmic units
- Reusable services such as Dem, NvM, and FiM

These methodologies are a reason that many OEMs and suppliers have decided to utilize AUTOSAR as their ECU architecture to aid in the adoption of ISO 26262. The following paper aims to highlight some of the AUTOSAR best practices when attempting to follow an ISO 26262–compliant process.

AUTOSAR Toolchain and Workflow Considerations

Use Simulink and Stateflow to model fault detection and software diagnostic functionality

The Simulink® and Stateflow® languages allow users to build up a variety of different types of algorithms. This same pattern holds true with AUTOSAR algorithms that are being modeled inside of Simulink and Stateflow. Simulink and Stateflow are often used to develop AUTOSAR application software components but are typically overlooked for high-ASIL functionality such as fault detection and diagnostics due to tool qualification concerns. To address this, MathWorks provides pre-qualification artifacts (artifacts that aid in tool classification), qualification artifacts, and a reference workflow for developing algorithms for all ASILs. Thus, there is no need to give up the benefits of a simulation environment when developing fault detection, maturation, degradation mode management, and diagnostic algorithms. To ease the diagnostic integration with AUTOSAR architectures, AUTOSAR Blockset has standard blocks that can implement Diagnostic Event Manager (Dem) interactions. In the below example, a `DiagnosticMonitorCaller` block is used. This block will simulate and generate code that is compatible with an AUTOSAR-compliant Diagnostic Event Manager.

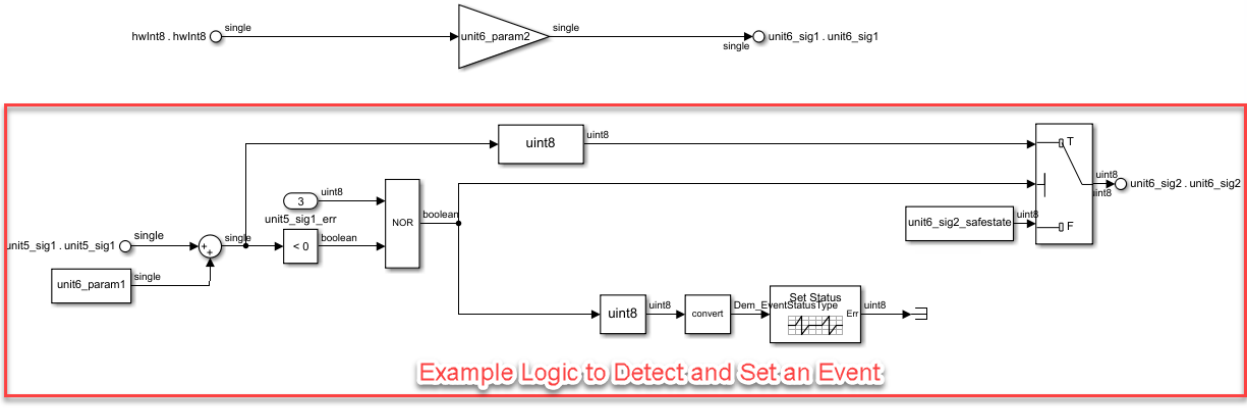

Figure 1. Example Simulink algorithm for fault detection.

Use standard AUTOSAR services (e.g., NvM, FiM, and Dem)

The AUTOSAR standard has defined several commonly used software services that most automotive ECUs need. For this example application, relevant services include Diagnostic Event Manager (Dem), Function Inhibition Manager (FiM), and Nonvolatile Memory Manager (NvM). Each service can be used interchangeably from one AUTOSAR application to another. It is advantageous to purchase these services from a COTS provider instead of creating a custom solution that will require additional verification work with possible reuse or compatibility issues.

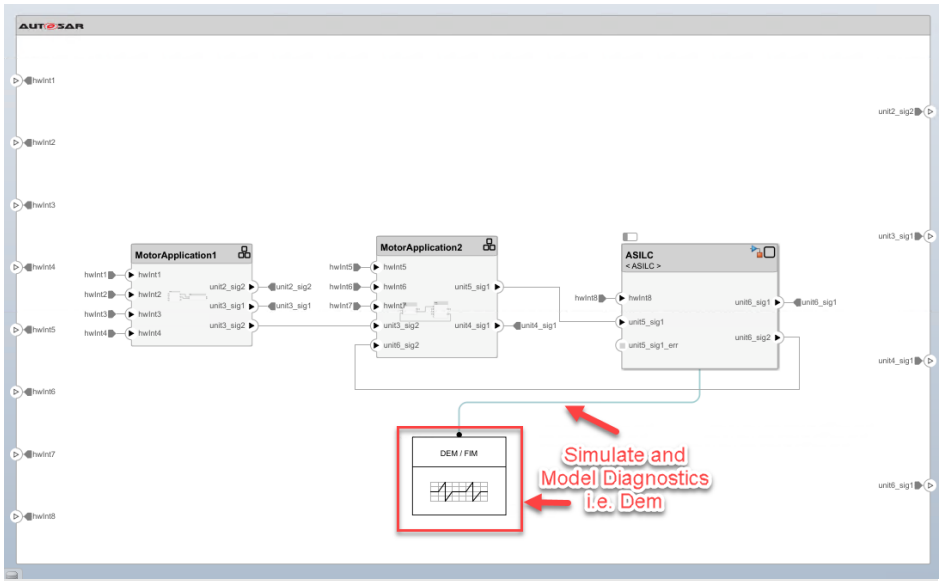

Figure 2. Example system-level Model with Dem RTE Block.

Simulink also makes using these services easy. AUTOSAR Blockset provides blocks that allow components to call these standard services with correct APIs in the generated code from Embedded Coder®. AUTOSAR Blockset also contains blocks with stub implementations of the services themselves. When used in a harness or AUTOSAR architecture model, these service provider blocks enable engineers to perform model-in-the-loop (MIL) or software-in-the-loop (SIL) testing without the complete service implementations.

Leverage prequalified/qualified ISO 26262 tools

If any automated tools are used in the creation of algorithmic content for an ISO 26262–compliant algorithm, those tools may need to be qualified. IEC Certification Kit provides the work products required for confidence in the use of Model-Based Design tools such as TCL (Tool Confidence Level), tool qualification artifacts, a reference workflow for Model-Based Design, and so on. As an example, an Embedded Coder reference workflow is shown below. The functional safety manager can leverage these artifacts and significantly reduce their tool qualification workloads.

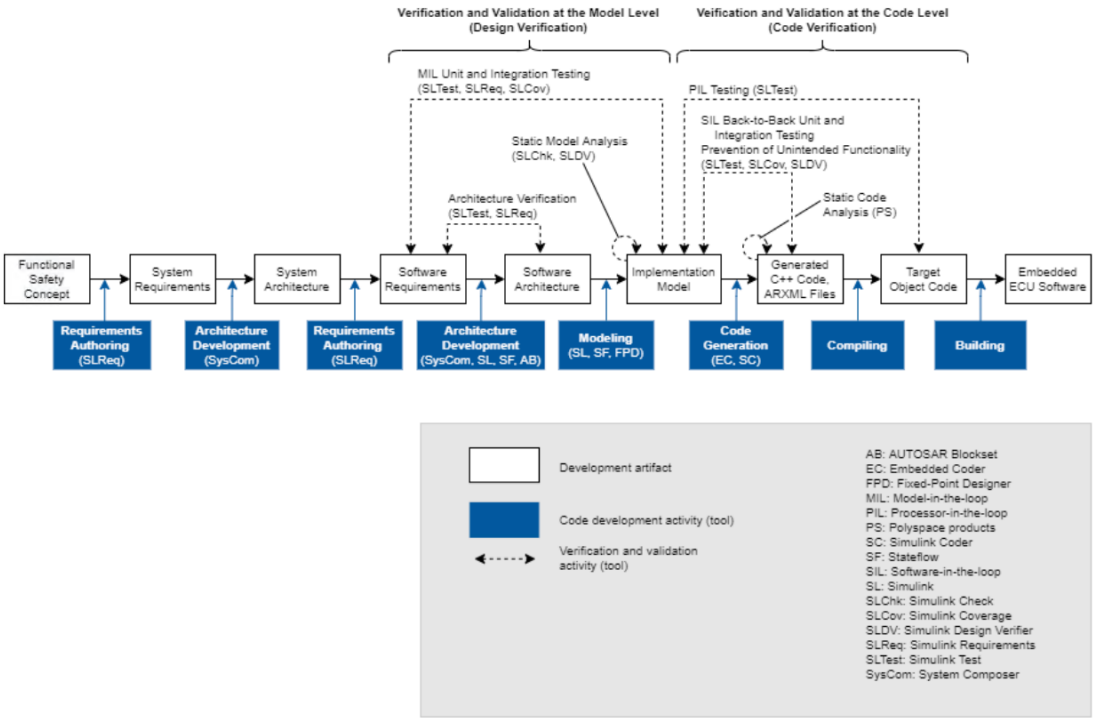

Figure 3. Reference workflow for Model-Based Design that complies with ISO 26262.

Utilize incremental testing strategy

A testing strategy that works well with ISO 26262 is to have an incremental, bottom-up testing methodology. This testing strategy starts from the unit level and works its way up through the various integration levels (shown below) until a full-software integration is tested:

- Unit-level MIL and SIL testing
- Unit-level static code analysis
- Feature-level MIL testing
- Application-level MIL testing
- Software integration–level SIL testing

The process starts with quick iterations of unit-level simulation. After these iterations give the development team a level of confidence, the activity shifts from model level to code level. In Simulink you can simulate using MIL and SIL tests with Simulink Test™, Simulink Coverage™, Simulink Design Verifier™, and Requirements Toolbox™. Embedded Coder is then used to generate the implementation. Back-to-back MIL/SIL testing can also be done to ensure that the simulation matches the behavior of the generated code. Once the unit-level test is complete, the first integration-level test can occur.

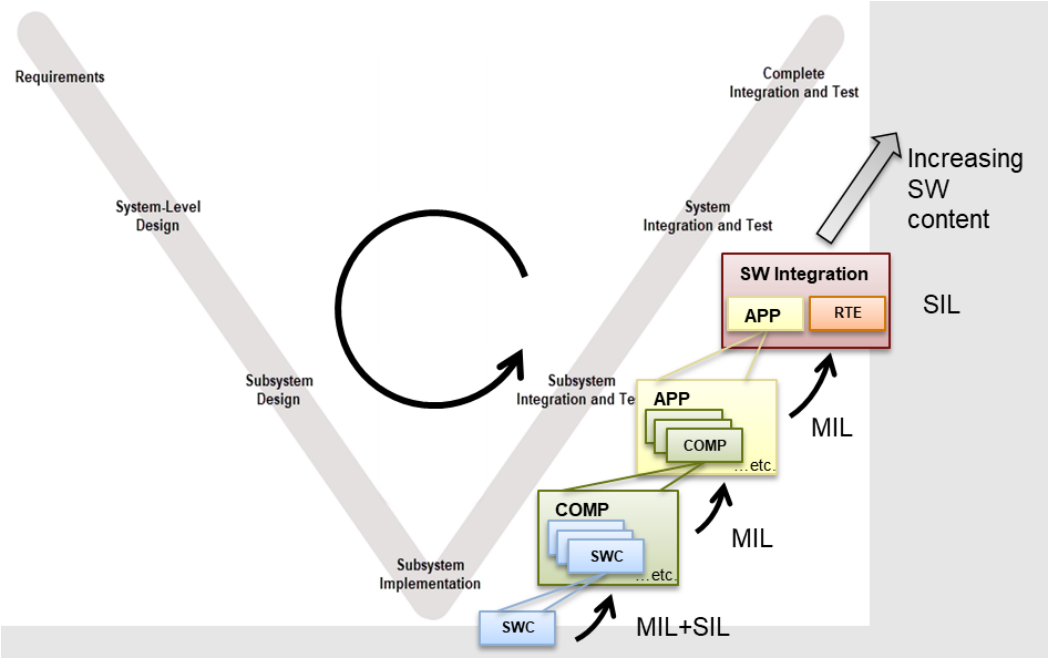

Figure 4. AUTOSAR incremental unit and integration testing strategy.

ISO 26262 Part 6, clause 10 prescribes methods for integration testing. This includes several supporting techniques such as requirements-based testing, interface testing, fault injection testing, and so on. These techniques need to be used on the software that will be deployed in production. In AUTOSAR, this consists of application software, basic software, and auto-generated integration code (RTE layer). The following diagram illustrates this software partitioning, where the MATLAB® membrane indicates portions generated from Embedded Coder.

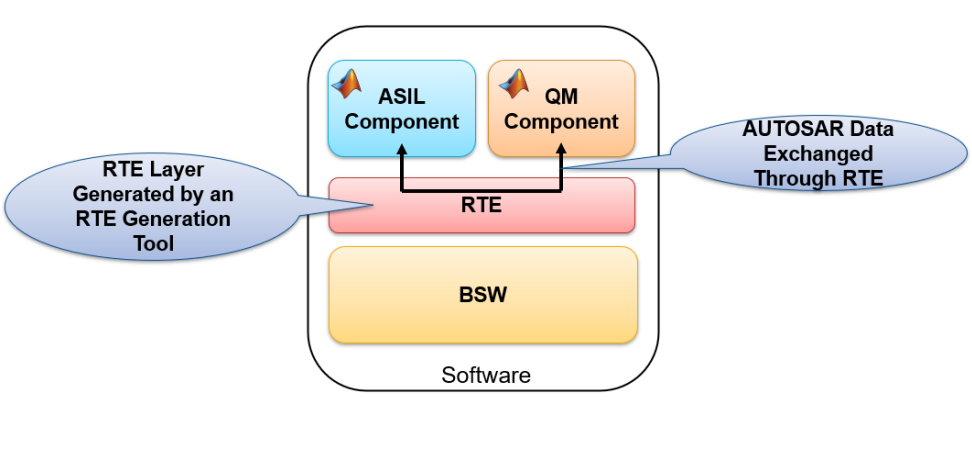

Figure 5. AUTOSAR software architecture.

Even though formal integration testing needs to be done on the actual target, it is highly recommended to utilize simulation early in the design cycle for integration checks. In Simulink,

it is easy to create AUTOSAR compositions and test harnesses to exercise the different levels of integration. These checks can increase the probability that when you get to final integration-level testing on the target, any data exchange mismatches will likely have been caught. The goal of this step is to reduce the number of items that will be caught once the software is on the target. Figure 6 shows an example of an integration-level harness for simulation testing.

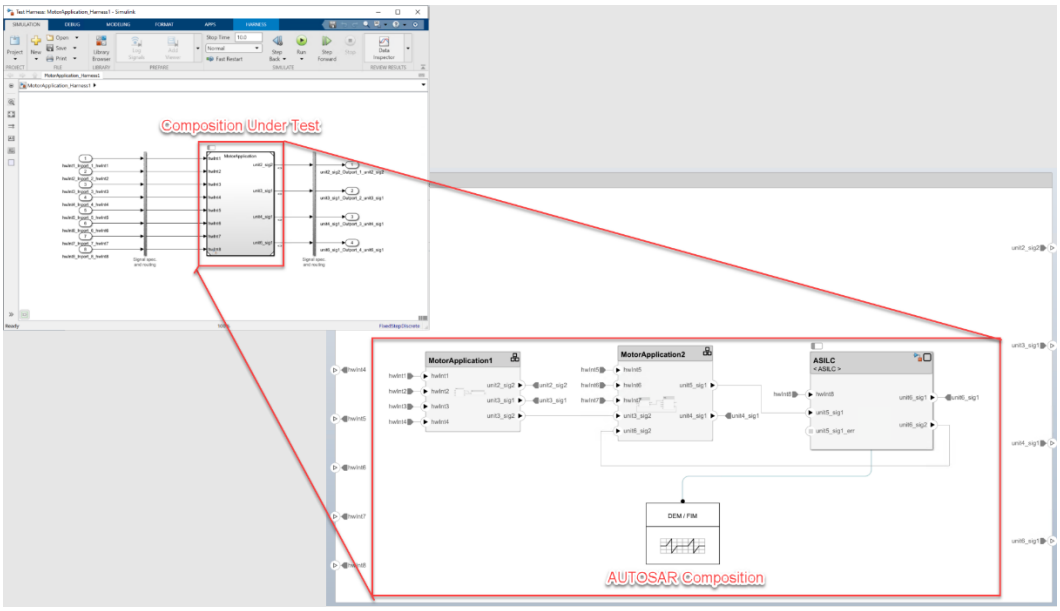

Figure 6. Integration simulation in Simulink.

Based on these factors, a formal integration testing strategy consists of:

1. Unit-level MIL and SIL testing on each AUTOSAR software component
2. Initial integration testing on AUTOSAR compositions in Simulink, with iterative simulations
3. RTE generation and full software build integration*
4. Software integration testing based on ISO 26262 requirements*
5. Static analysis using Polyspace

* Activities completed on final target hardware

AUTOSAR Architectural Constructs

Use software components as a unit boundary

ISO 26262 Section 6 goes in-depth on principles for the creation of an ISO 26262–compliant software development process for each ASIL. Unit-level and integration-level software activities are called out specifically. AUTOSAR, on the other hand, has compositions, atomic components, and runnables. Thus, it is necessary to use AUTOSAR architectural constructs in a way that is compatible with the language of ISO 26262. Individual entry-point C functions are mapped to runnables, which are encapsulated in atomic software components. In Simulink, components map to a Simulink model, and runnables map to either different rates in the model, function-call subsystems, or Simulink function blocks. Related software components are assembled into compositions representing individual software features, which are further

assembled into compositions representing subsystems and systems. AUTOSAR compositions are fully virtual—there is no code representation of a composition.

Atomic software components are a strong candidate for the ISO 26262 software unit definition for several reasons:

- Atomic software component elements and interfaces are mappable at the model level.
- Simulink model boundaries are ideal for creating test harnesses and test cases.
- Content that is graphically in a model will generate into an encapsulated set of .c and .h files.
- Runnables within a component should be highly coupled and are best tested as a unit.

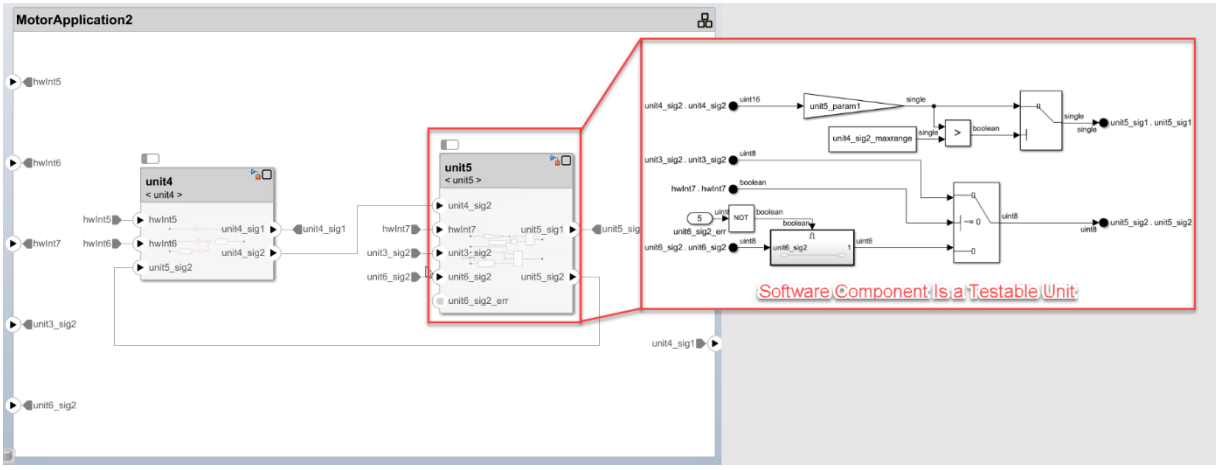

Figure 7. Example AUTOSAR software component as a unit boundary.

Segment like ASIL components into grouped compositions

ISO 26262 allows for an ECUs application layer to contain units of different ASIL if freedom from interference can be demonstrated. If an application needs to have multiple ASILs, it is recommended to segment these software components into distinct grouped compositions. For example, one composition could be dedicated to components that are ASIL-D–rated and another composition could be dedicated to ASIL-B–rated components. The reason for segmenting the algorithm in this manner is to clearly see which components belong to each ASIL. This also allows the validation and verification process to be consistent for each ASIL. Integration testing can be done at the rigor required for each ASIL.

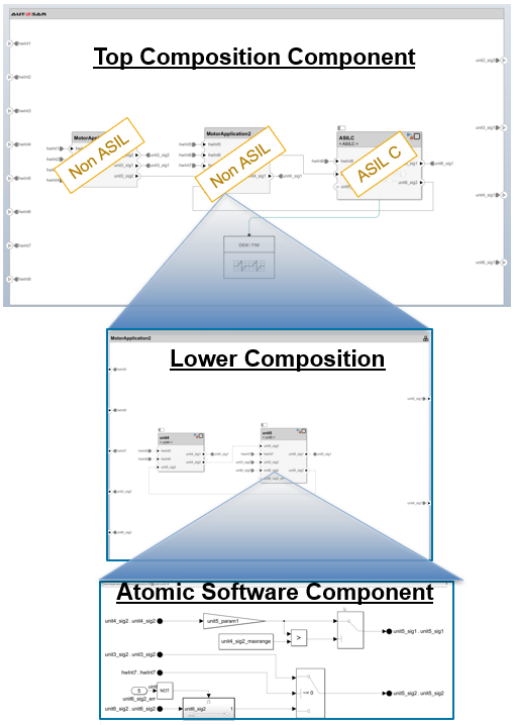

Figure 8. Software architecture with divided ASIL compositions.

Software address methods to segment AUTOSAR components into memory sections

ISO 26262 provides example safety mechanisms and measures to achieve freedom from interference. Safety mechanisms take into consideration example representative faults regarding:

- Timing and execution
- Memory
- Exchange of information

One method to alleviate memory and the exchange of information faults is by using different memory sections for each ASIL. This is easily achievable using the AUTOSAR concept of software address methods. Software address methods act as an abstraction layer on top of implementation-specific memory sections. An architect can tag various AUTOSAR elements such as runnables, ports, and parameters with different software address methods. Then, the architect can target these software address methods to different memory sections on the microcontroller. The software address method can easily be selected and viewed in the Simulink model also. We recommend defining a different set of software address methods for each ASIL to enable the most granularity for memory partitioning.

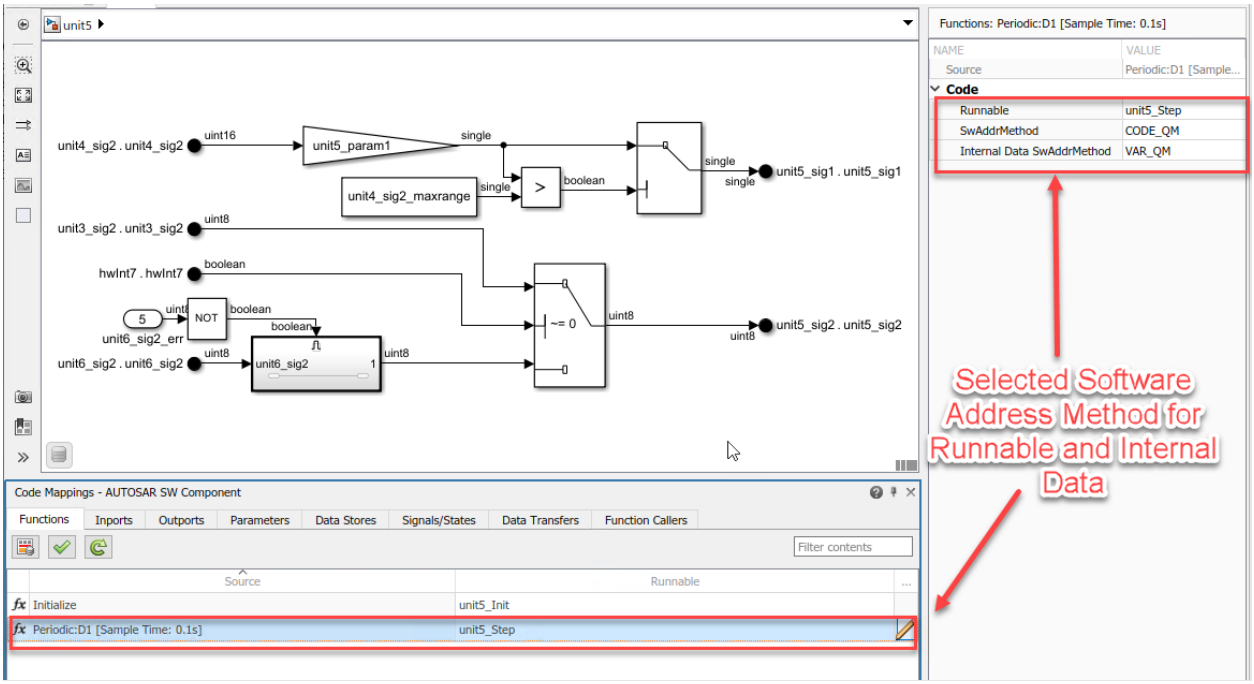

Figure 9. Usage of software address methods for runnable and internal data.

Rearchitect existing code base if a transition is made from non-AUTOSAR to AUTOSAR

When an organization decides to adopt an AUTOSAR architecture, this is often a migration from a previous non-AUTOSAR code base. For many organizations, it is tempting to simply take the existing algorithm and wrap it in AUTOSAR constructs. An example of this would be the use of a complex device driver to wrap a large section of existing legacy C code. This is not an advantageous method to utilize when switching to an AUTOSAR architecture. It will be more advantageous to view this as an opportunity to ensure that your organization's architecture is thoughtfully constructed and will meet your organization's functional safety goals. Therefore, it is imperative to have a software architecture review and adjust the organization's architecture considering available AUTOSAR constructs and constructs that will help your organization reach your functional safety goals. However, for suppliers that need to support a one-off AUTOSAR program with an existing software architecture, the complex device driver approach might prove more feasible than the development effort required for a rearchitecture.

Define a data management strategy

While many viable data management strategies exist, targeting AUTOSAR constructs with AUTOSAR Blockset helps to narrow this topic. Data that is scoped to the component level should reside in the model workspace. This facilitates code-mapping workflows and prevents unintentional sharing of calibration and measurement definitions. The model workspace can be configured to reference an external M-file or MAT-file as a data source to manage value sets separately from the model files. Each component should store global objects that they reference, such as data and data types (e.g., Alias, Bus), in a component-specific data

dictionary. If preferred, types shared and managed between many components or teams can be grouped and referenced from a common data dictionary. In integration, composition models help enforce consistency across components, as any duplicate definitions across data dictionaries in the model hierarchy must be identical. Alternatively, merging all data dictionaries (while validating entries) into a single data dictionary at integration time can accomplish this as well.

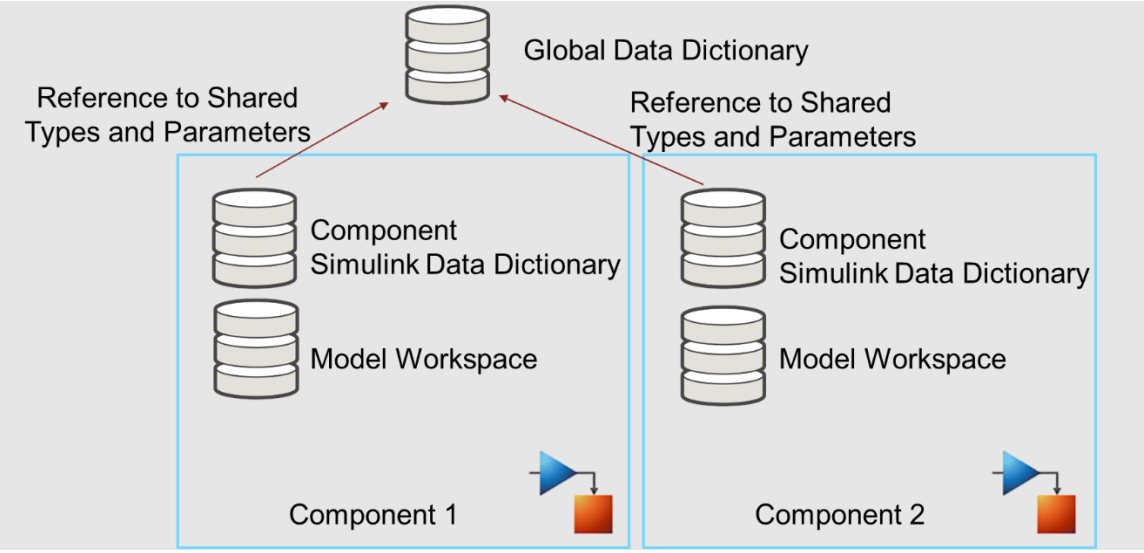

Figure 10. Example AUTOSAR data hierarchy in Simulink.

AUTOSAR Data Transfer Mechanisms

Use the RTE to pass safety-critical data

AUTOSAR is an intentionally layered software architecture. At the lowest level, there is Basic Software (BSW). The next layer above, the BSW, is the run-time environment (RTE). The top layer is the application layer (ASW). All AUTOSAR port data is passed through the RTE and managed by the RTE. This allows for the RTE to add additional layers of protection, such as managing specific memory sections for data and error statuses to detect transmission errors. High-integrity and safety-critical data should be passed through the RTE to benefit from this additional protection. In addition, the interfaces between these components can be clearly defined and managed in the authoring tool at the virtual function Bus level, abstracting away the mechanism of data transfer of intra-ECU and inter-ECU.

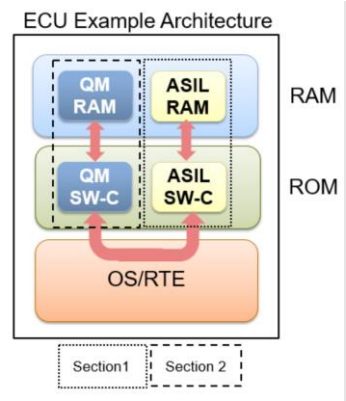

Figure 11. Software components transfer data through the RTE.

Use implicit data transfer for high-integrity signals

AUTOSAR has several data access modes when defining data transfer between runnables and components of the software and electrical system. The two main types of access for data elements in AUTOSAR ports and inter-runnable variables are explicit access and implicit access. Each data access mode has its own benefits. Implicit transmission ensures that each runnable using a piece of data will access an independent copy of the data, decoupled from all other readers and writers of the data. The RTE maintains a source of truth for the data element, which is written to the runnable copy right before the runnable begins execution, and is updated from the runnable copy right after the runnable finishes execution. Therefore, if the runnable is interrupted, the runnable will still have the previous copy of data when execution resumes. Also, other runnables will not see any intermediate value computed by the interrupted runnable until the interrupted runnable finishes execution. This ensures that the data does not change nondeterministically during a single time step. Explicit data transmission, on the other hand, will always give the runnable the current value that has been set for the data element by the provider at the time of the request.

For signals that need to have high integrity, it is recommended to use implicit data transfers for these data elements. This ensures that the consumer will have a nonvolatile copy of the data to operate on while performing calculations. In Simulink, this can be configured at an individual data element level by changing the transfer mode from explicit to implicit.

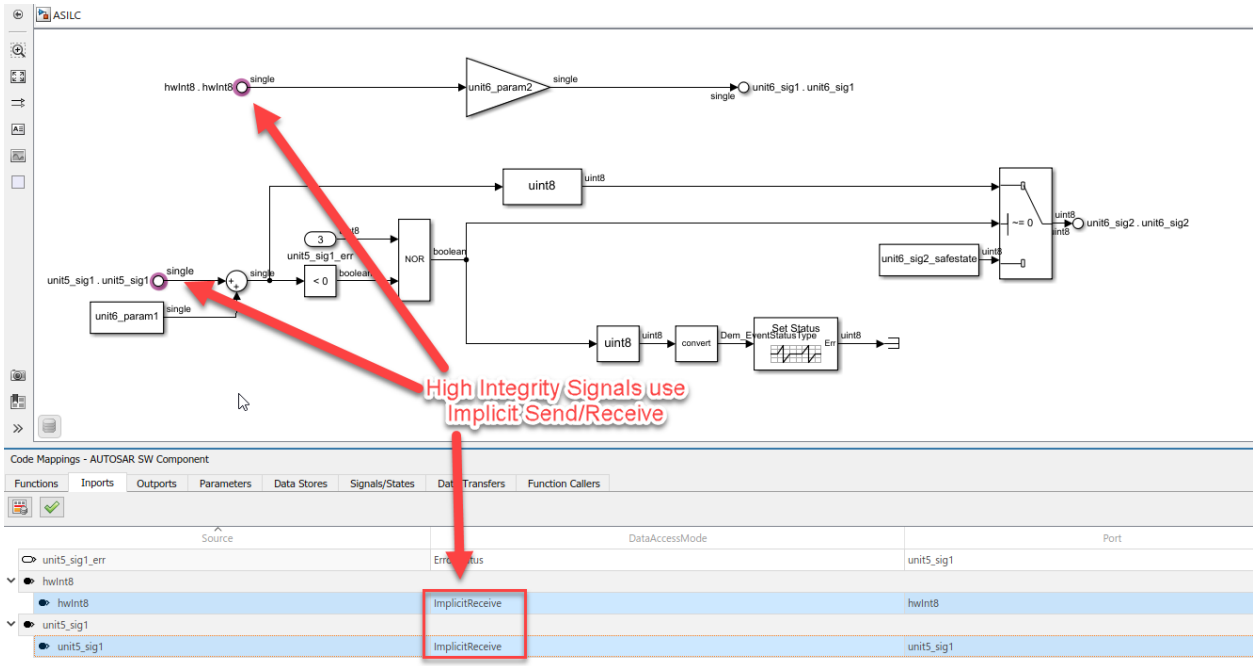

Figure 12. Configure implicit send/receive at the component level.

Use port error status when transferring data between different ASIL components

AUTOSAR can provide any runnable with an error status for each data element. AUTOSAR defines quality-of-service attributes, such as `IsUpdated` and `ErrorStatus`, for sender-receiver interfaces. The `IsUpdated` attribute allows an AUTOSAR explicit receiver to detect whether a receiver port data element has received data since the last read occurred. When data is idle, the receiver can save computational resources. The `ErrorStatus` value is intended to catch various failure modes in the data integrity. This can notify the consuming application software if the data is received and can be trusted or not trusted. It is recommended to use the error status port in application software components for any data signals that require high integrity, such as when exchanging data between sections of software with different ASIL.

This error status can be configured in Simulink for each data element by constructing a new port and matching it to the data element port. This allows the algorithm to have two Simulink ports. One port is used for the data value and the other for the error status.

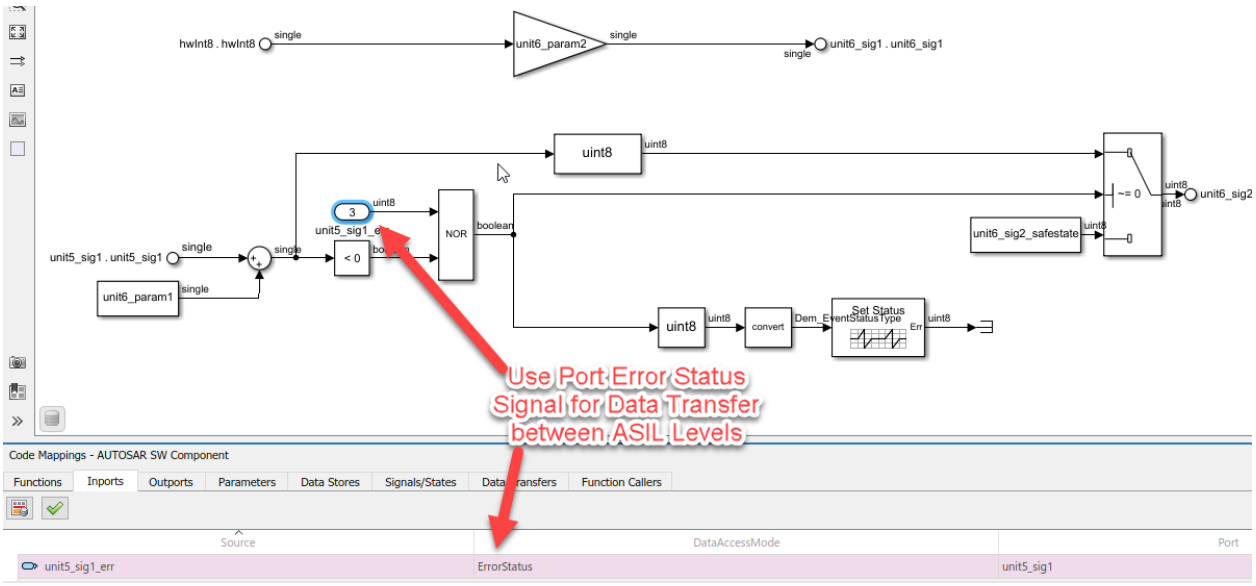

Figure 13. Configure port error status.

Use end-to-end protection

End-to-end protection is a type of protection provided by the RTE implementation and is intended to detect data transmission errors from one software component to another through an AUTOSAR port. When using end-to-end protection, the returned `ErrorStatus` will differ to reflect the potential failure modes that can be caught with end-to-end protection. The software application should check these values and decide how to respond to these types of failures. The transmission of the data through these ports can either be over a medium such as CAN communication or through a RTE call between two components on the same ECU. End-to-end protection can be implemented for any AUTOSAR port. We recommend using end-to-end protection for any high-integrity AUTOSAR data that goes over a communication channel such as CAN, LIN, or Ethernet.

End-to-end protection can be defined in the AUTOSAR architecture tool. This setting can also be managed and viewed in the Simulink model. Figure 14 explains how to tag a data element in Simulink with end-to-end protection.

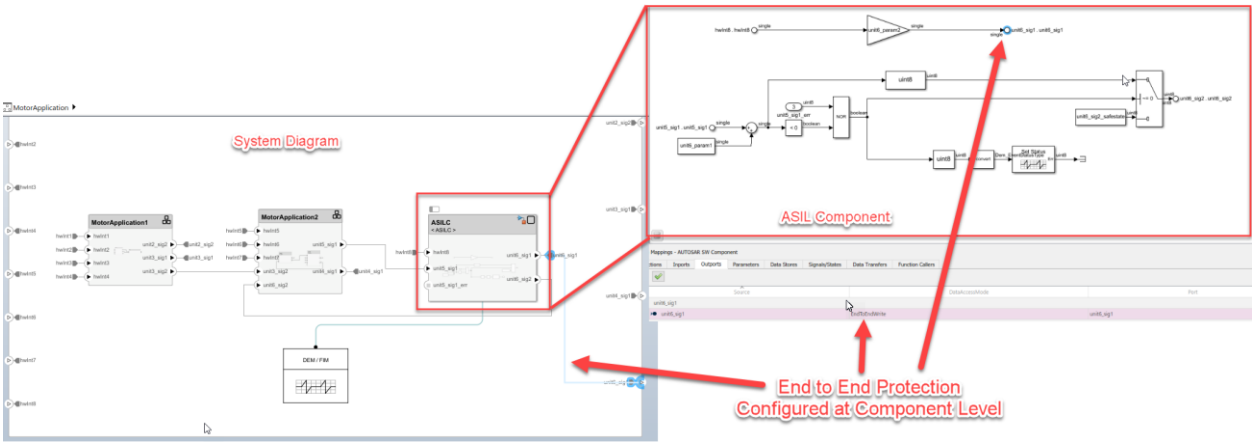

Figure 14. End-to-end protection configuration at the component level.

Summary

The purpose of this paper has been to investigate ISO 26262 and AUTOSAR best practices. AUTOSAR was developed to be complementary to ISO 26262 and provide constructs that can ease adherence to ISO 26262. The best practices in this paper are AUTOSAR constructs and workflow items that can be leveraged in Simulink to help with ISO 26262 adoption in:

- The AUTOSAR toolchain and workflow
- AUTOSAR architectural constructs
- AUTOSAR data transfer mechanisms

The best practices explored in this paper are also complementary to the white paper, "10 Best Practices for Deploying AUTOSAR Using Simulink." These best practices should be reviewed and used when deciding on your organization's workflow for ISO 26262. In addition, a development organization should narrow down which AUTOSAR constructs should be the recommended standard within the development organization. MathWorks consulting has helped numerous customers set up an ISO 26262–compliant workflow and construct a strategy for their AUTOSAR architecture on the basis of qualified use cases of Model-Based Design.

Next Steps

MATLAB and Simulink for AUTOSAR

mathworks.com/solutions/automotive/standards/autosar

Contact MathWorks Consulting

mathworks.com/services/consulting.html