Functional Safety Working Group

White Paper

December 13, 2023

Contents

I. Introduction

The Accellera Functional Safety Data Model is intended to support the generation and interchange of Functional Safety Content that represents diverse elements of the safety cases of safety-relevant systems, modules, components, and IP in related industries. The data model is a foundational component to complete the working group objectives defined in the Functional Safety Working Group White Paper [1]. The goal and scope of the data model is to capture and propagate the Functional Safety (FS) content across the different safety operations and the distributed development environment, from system to IPs. Achieving this goal will enable automation, interoperability, and traceability across safety activities.

In a distributed development environment with multiple organizations as suppliers and customers (integrators), it is efficient to perform the safety activities separately at each level or in each organization or team. The safety activities (operations and the resulting work products) involved in these developments for activities within a single organizational layer are depicted in Figure 1. The interchange between organizations at different "layers" is depicted in Figure 2. Throughout this paper, the term "Intra-layer" and "Inter-layer" will be used and the definitions for those terms are as follows:

- **Intra-layer**: Through different safety analysis/operations of the same hierarchy level, e.g., FMEDA analysis, verification
- **Inter-layer**: Between layers of design hierarchy/supply chain, e.g., System ↔ Module ↔ Component ↔ IP

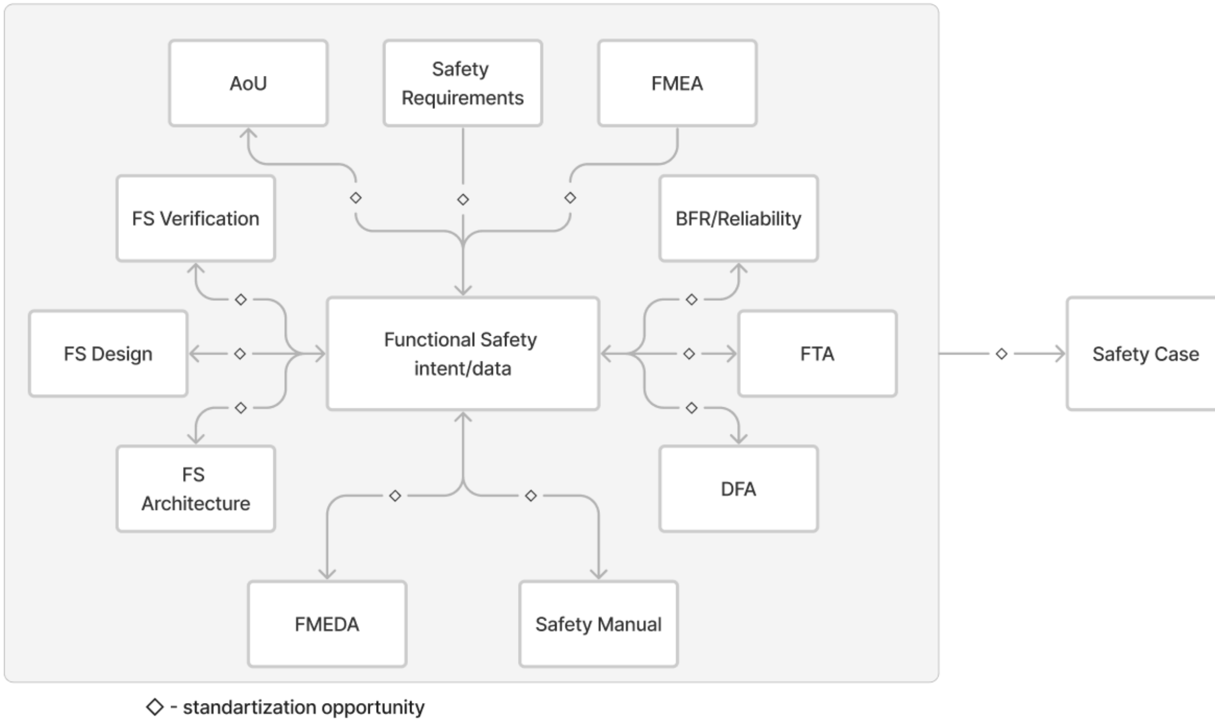


Figure 1. Representation of the concept of the data model to cover the intra-layer operations and work products.

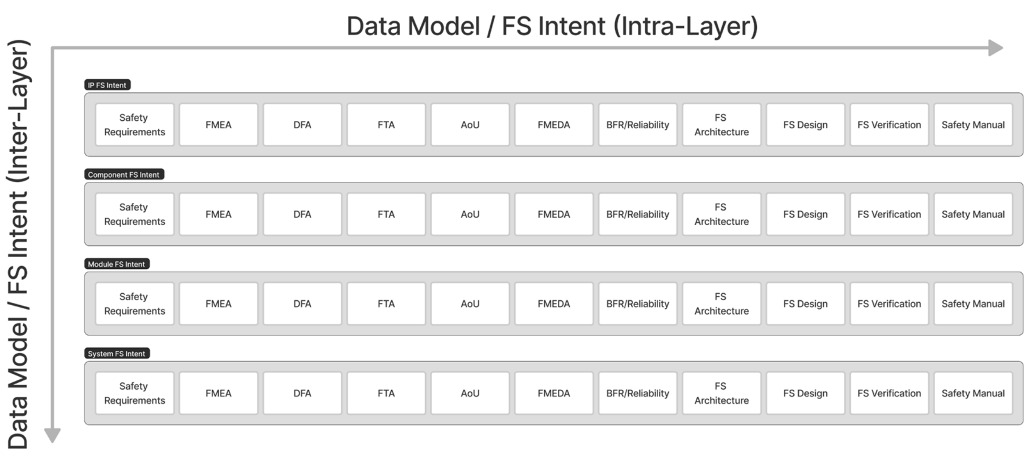


Figure 2. The data-model includes information to allow exchange for both intra-layer and inter-layer requirements.

The initial focus of the Accellera working group is to develop a data model supporting FMEDA (Failure Modes Effects and Diagnostic Analysis) creation and exchange within the following scope:

- Domains (Digital, Analog, SW)
- Industries (Automotive, Industrial, Machinery)
- Supply Chain layers (IP, Component, Module, System)

The metrics specific to FMEA (Failure Mode and Effects Analysis) are not directly covered in this white paper, however support can be included easily in a subsequent version as the rest of the data model is shared between FMEDA and FMEA. The Accellera FS Data Model support for FS content related to architecture, requirements, FTA, DFA, verification and validation (V&V), and others will be completed at a later stage of development.

The data model implementation supports two main use cases:

1. FMEDA evaluation: A safety analysis is performed and described, for example, by using a command-based formalism describing the atomic actions (e.g., create the safety analysis, create a failure mode, etc.). When the user decides to generate final reports, all of the outputs are also stored in the data model. In this use case the provided authoring information is evaluated with the intent to populate the data model and to be able to generate final reports.
2. "As is": A safety analysis is shared "as is," as for example an FMEDA table or summary. In this use case there is no authoring information but only failure rates and metrics to be exchanged as outputs (for example, following a numerical evaluation of the data model) or imported as inputs.

As stated in the Accellera FS WG white paper [1], the goal for the Accellera FS standard is to work in alignment with well-established safety standards (e.g., ISO26262 [2] and IEC61508 [3]) and to facilitate their implementation. Hence, calculations and definitions are meant to be consistent with such standards (unless stated otherwise).

Figure 3 describes the phased approach used by the Accellera FS WG to develop a functional safety language:

- First, the process of Functional Safety Analysis is formalized.
- Second, the conceptual data model will emerge from the data exchanged and the operations formalized in the first step.
- Third, the Functional Safety language will be derived formally from the conceptual data model.

This paper will cover the first and second step, while the third will be part of the Language Reference Manual (LRM) to be published later and will constitute the Accellera Functional

Safety Standard. A sample language will be deployed in this paper solely for the sake of illustration through examples, however the final standard might differ.

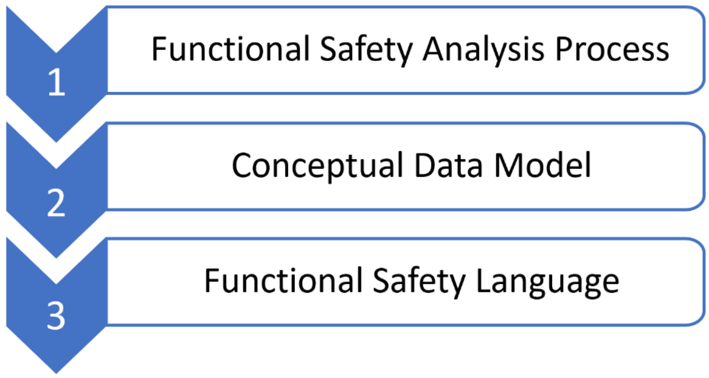


Figure 3. The Development process of the Functional Safety Language.

This white paper is organized as follows:

- **FMEDA Process** captures the formalization of the process to perform a Failure Mode and Effects and Diagnostic Analysis (FMEDA). This is a conceptual representation that identifies the elements of the FMEDA process (e.g., FS Hierarchy, Failure Modes, Technology elements) and how they are connected to each other
- **Design Representation and Mapping of Data** connects the FS data model with the design representation and details the concept of mapping. Mapping is used to connect different sets of data (e.g., the Functional Safety hierarchy to the design hierarchy) and therefore several different mapping types are defined.
- **FMEDA type** introduces the definition of distinct types of FMEDA: calculation-based and assumption-based. These concepts clarify how the design metrics are extracted or provided to calculate the failure modes distribution.
- **Conceptual Data Model** summarizes the basics of an entity-relationship data model, continues with general considerations about the data model and connects the elements of the FMEDA process identified in FMEDA Process above to the entities of the FS data model. It then expands the entities covered to include all attributes that constitute the complete and detailed FS data model.
- **Detailed Annotations on the Data Model** includes several detailed discussions about the methodology supporting the definition of some of the data model attributes, based on the FMEDA process.
- **Annex A: Data model** expands the entities defined in Conceptual Data Model above to include all attributes that constitute the complete and detailed FS data model.

- **Annex B: Language** covers a prototype language, which is used to illustrate examples of FS projects created using the defined data model. Additionally, hypothetical language constructs are covered to illustrate additional opportunities that are available in the scope of this work.
- **Annex C: Add-on to v0.1** reports additions to the proposed data model/language that will be considered for inclusion beyond the first release.
- **Annex D: Repository** includes several examples created using the prototype language, ranging from a one-picture example to a step-by-step illustration accompanied by source code, author's comments and equivalent FMEDA tables.

II. FMEDA Process

The FMEDA process is a bottom-up, inductive analysis describing how elements of a system can fail, and how the effects of defined failures can be mitigated (detected or controlled) to maintain a safe state. The remainder of this section details the traditional FMEDA process to identify the data and operations that will eventually lead to the definition of the FS data model.

The traditional components and operations of an FMEDA are listed below and highlighted in Figure 4:

- The process receives as input a representation of the **Design Under Analysis** (DUA) via a comprehensive list of all of the components in it, typically organized in a hierarchy as appropriate.
- An analysis of the intended functionality identifies the functional safety analysis hierarchy (**FS Analysis Hierarchy**) detailing the portions of the design that are safety related and which have the potential to violate a safety goal or safety requirement.
- For each relevant portion of the FS Analysis Hierarchy, the Failure Modes analysis defines the Failure Mode (FM) hierarchy (**FM Hierarchy**) through enumeration of the possible failure modes of each element that can cause its failure to function as specified (malfunction).
- One or more technology elements are identified for each Failure Mode of the DUA, based on the technology elements available (**Technology Elements Library**).
- The Failure Mode Effects Analysis (**Failure Mode Effects** or **FME**) identifies the effects of the FM on the DUA as seen when instantiating the DUA into the next level of the supply chain.
- Safety diagnostics are selected from the inventory of potentially available safety mechanism(s) (**Safety Mechanisms Library**) and applied to mitigate the identified failure modes and bring the system to a safe state within the required time.

The Safety Mechanism may be implemented within the design under analysis or in another element of the system into which the component is integrated. Safety Mechanisms in elements outside the scope of a safety element provided by a supplier are documented in the element's safety manual as Assumptions of Use (AoUs).

The formalization of the process to perform an FMEDA captured in Figure 4 identifies the elements of the FMEDA process and how they connect to each other. These form the basis to define the categories of information of the FS data model and how they are related. The next section details these relationships.

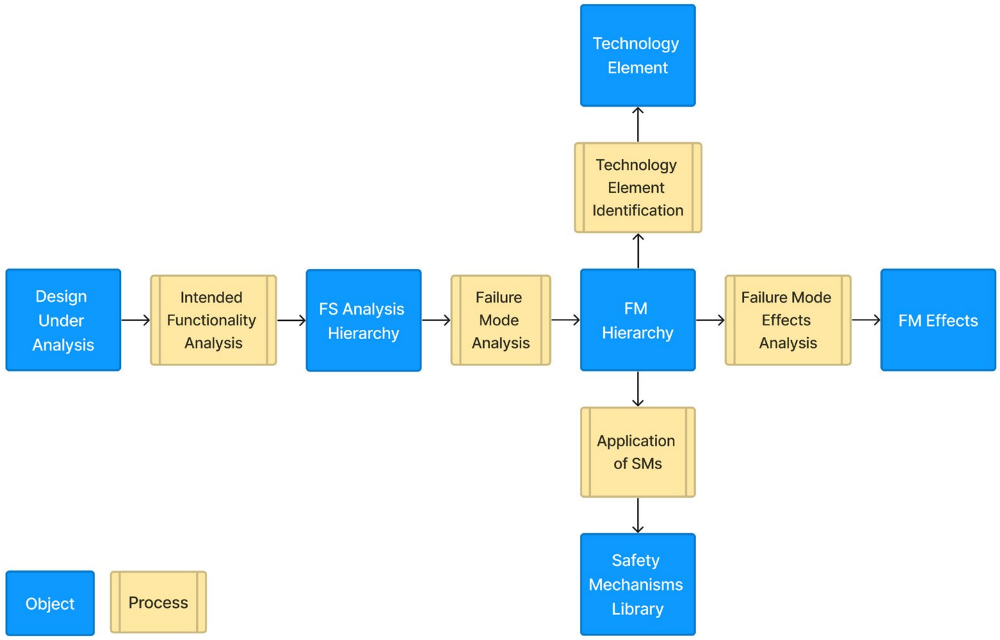


Figure 4. Fundamental data and operations in the creation of an FMEDA.

III. Design Representation and Mapping of Data

This section introduces the concepts of design representation and design mapping, both of which are fundamental to the formalization of a data model supporting and automating FMEDA activities.

A. Design Representation

A design can have multiple representations as it matures through its development lifecycle, and safety analysis can be performed on any representation within the selected scope (i.e., IP, Component, Module). It may be convenient for the analyst to organize the elements of the source design in abstract or functional groupings to ease the identification of failure modes and their associated safety mechanisms. This is permitted if the representation is complete with no omissions in scope.

In a functional partitioning, the design is represented by a hierarchy of functions and subfunctions with their interfaces and interactions. In this representation, specific implementation details such as target technologies may have not been relevant. In a structural design representation, the actual intended circuit implementation is displayed. Figure 5 and Figure 6 show an example of a functional representation and a structural representation, respectively [15]. Note that the concept of functional and structural design representations applies to all layers of the supply chain (IP, Component, Module, System), which is a distributed development environment with multiple organizations as suppliers and customers (integrators).

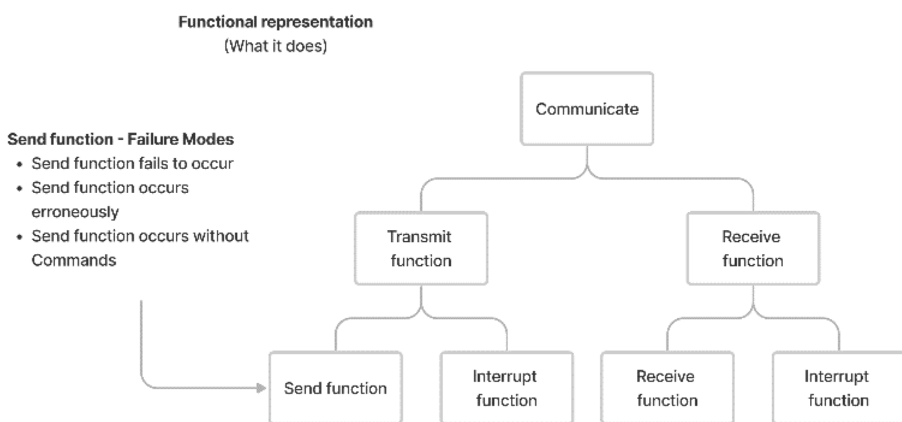


Figure 5 Functional Representation of a component

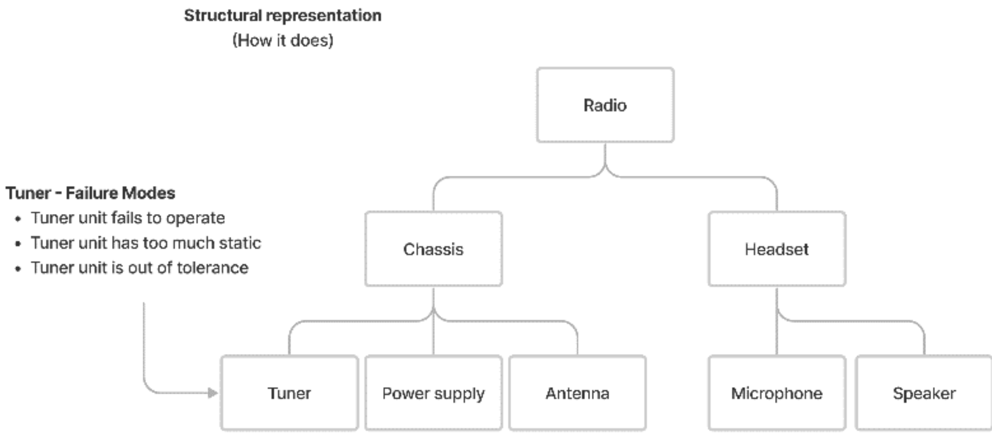


Figure 6 Structural representation of a component

The data model supports (and is agnostic) of whether the design representation is functional or structural.

Examples of common products and design representations across the supply chain are included in Table 1.

Table 1. Description of layers and common design representations at each layer.

Supply Chain Layer	Layer definition	Product examples	Common design representations
System	Captures the function(s) visible at the driver/vehicle level	ECU(s) including sensing, processing, automation	System Model
Module	Implements one or more safety goal(s) and can be shared by different systems	• Sensor, actuator, processing module • PCB(s) and enclosure	System Model
Component	Implements one or more safety function(s) and can be shared by several modules	• Packaged part (die + package): - SIP/MCM: System-in-package - IC: Integrated circuit - Simple components: Passives • Die-level: - Single function: e.g., ADC - SoC: System-on-chip: multiple functions/ subsystems (e.g., processor, peripherals, accelerators,	Block Diagram, specification, Modeling language (SysML [10], IP-XACT [8]), RTL, Gate-level Netlist [5]

		interface ports)	
IP	Implements one or more standalone (safety) function(s) and can be shared by different components	• Soft IP (e.g., SPI port, DDR controller, ML subsystem) • Hard IP (Analog function, e.g., MIPI PHY) • Foundation libraries (pads, memory array compilers, cell libs)	Block Diagram specification, Modeling language (SysML, IP-XACT), RTL, Gate-level Netlist

The design representation exists independently and is an input into the functional safety analysis to be performed. The independence of the design representation is critical in supporting the various scenarios encountered performing FS analysis within and across the supply chain. To perform the FS analysis, a connection to the design representation is commonly enabled through a set of mapping operations as explained in Mapping.

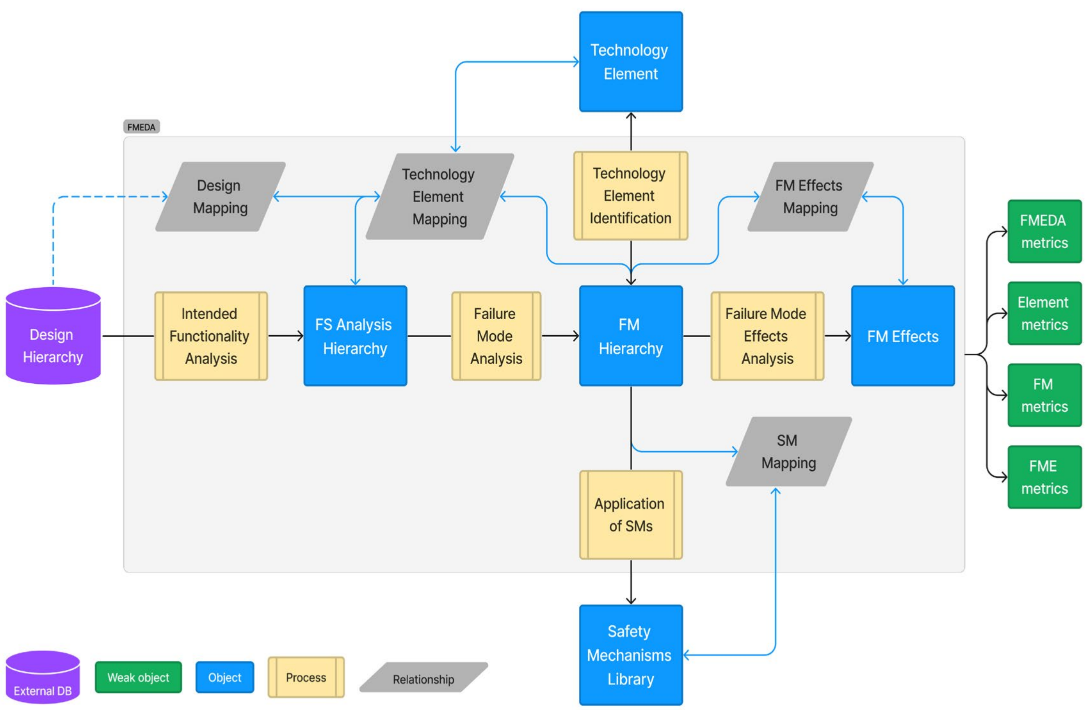


Figure 7. FMEDA process overview with mapping included.

B. Mapping

Mapping is defined as the operation of connecting one set of data to another. In the context of the Accellera Functional Safety Data Model, mapping connects data across the FMEDA workflow defined in Figure 7. Several types of mapping are defined in the context of performing an FMEDA, depending on the data it connects. The complete list is defined in Table 2 and is detailed in the remainder of this section.

Table 2. Data mapping involved in the FMEDA process.

Type	Description
Technology Element Mapping	Maps the FS Hierarchy elements and/or the Failure Modes to the Technology Elements
Safety Mechanism Mapping	Links a Safety Mechanism to the Failure Mode to protect
Failure Mode Effects Mapping	Maps failure mode(s) to a failure mode effect that can be exported to higher analysis levels
Design Mapping (FS Analysis Hierarchy)	Through the Technology Element Mapping, connects the design component(s) to the Functional Safety Hierarchy (represented as FMEDA Elements) identified during the analysis of the intended functionality
Design Mapping (Failure Modes)	Through the Technology Element Mapping, connects the design component(s) to the Failure Mode(s) they may cause

To illustrate the mapping concepts, the OpenRisc 1200 (OR1200) structural hierarchy will be used as a demonstration vehicle.

Design Mapping

Design Mapping connects the design instances of the source design to the FS analysis hierarchy, represented for example as Parts and Subparts, and/or to the Failure Modes. No specific design representation is assumed for the source design.

Mapping real design information to a given safety hierarchy supports mainly three purposes:

- Provides a simplified system partitioning targeting the failure mode definitions and safety hierarchy elements. Typically, safety engineers want to have the possibility to describe FS hierarchy elements and failure modes by using a simplified partitioning compared to the real design hierarchy, but still having a link with real design information.
- Having a link with real design information enables automatic computation of the failure rates assuming they will be computed according to the related mapped technologies and areas (e.g., number of transistors) that are evaluated following the design mapping.
- By providing a bidirectional mapping between the safety hierarchy and the real design hierarchy it will be possible to perform cross-checks, for example to verify the consistency of the technology mapping, potential overlaps of the design information mapped to the failure modes, and hierarchical inconsistencies.

It is important to note that the definition of the FS hierarchy is not required to align to the design hierarchy. The definition of the FS hierarchy is left to expert analysis. Mapping is subsequently performed connecting the defined FS analysis hierarchy to design components/instances. Figure 8 shows an example of such mapping where the subpart DCACHE RAM is mapped to a single design component such as a module instance, while the subpart MMU is mapped to a collection of instances, leaf cells, or other design structures.

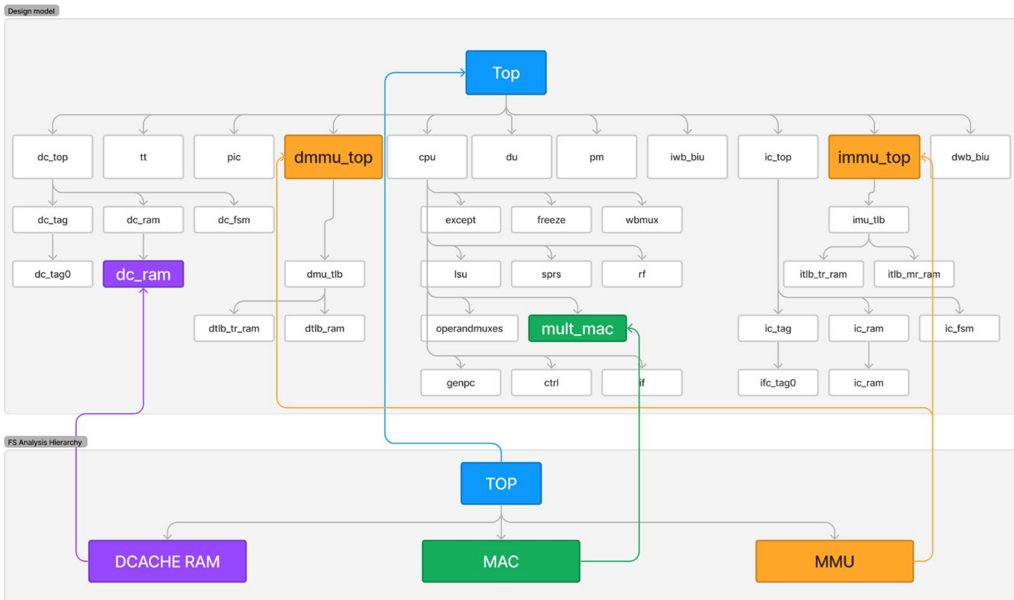


Figure 8. Mapping of functional safety hierarchy to design components.

The proposed data model does not support direct mapping of failure modes or elements to design objects. This is done through mapping of failure modes, elements, and technology elements. A "relationship" object that establishes such connectivity has a set of attributes that describes external design information. See details in Figure 21 and Figure 7. A data model-accurate example of an FMEDA project with source code examples is shown in Example 4. In this chapter a detailed description of mapping operations through various objects is omitted for simplicity.

Failure Modes Mapping

After the functional safety analysis hierarchy is set up, a failure mode analysis is performed. Failure modes are identified as functional safety hierarchical elements and represent unique objects within the data model, and there is link between a functional safety hierarchy element to the failure modes for that element. At this point, the design mapping may also be used to connect the design instances of the source design to the Failure Modes. Extending the prior example, two failure modes in Figure 9 are defined for the "MAC" while a single failure mode is defined for the "MMU" and "DCACHE_RAM." A failure mode can be mapped to a single design component such as a module instance (e.g., FM1_dcache), or a collection of instances, leaf cells, or other design structures (e.g., FM1_mmu). Also, more than one failure mode (e.g., FM1_mac and FM2_mac) can be mapped to a single module instance. Like the FS analysis hierarchy, a design mapping facilitates higher levels of automation such as automated failure mode distribution.

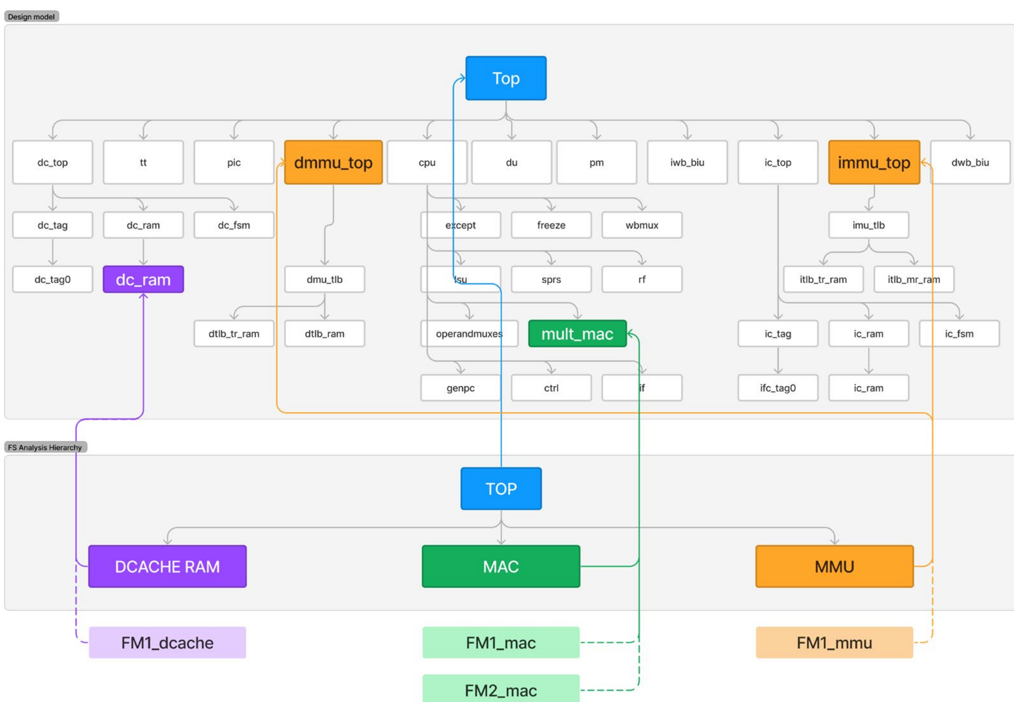


Figure 9. Mapping of failure modes to design components.

Safety Mechanism Mapping

Once the failure modes are identified, the safety features working to control or avoid failures to those failure modes can be defined. The safety mechanisms mapping connects the safety feature(s) deployed to detect and mitigate random failures, preventing the violation of a safety goal/requirement to the failure modes. A failure mode may be "protected" by one or more safety mechanisms, and similarly multiple failure modes may be "protected" by a single safety mechanism. Therefore, the data model supports a many-to-many relationship between failure modes and safety mechanisms. Figure 10 demonstrates these concepts by showing one-to-many, many-to-one, and one-to-one safety mechanism configurations.

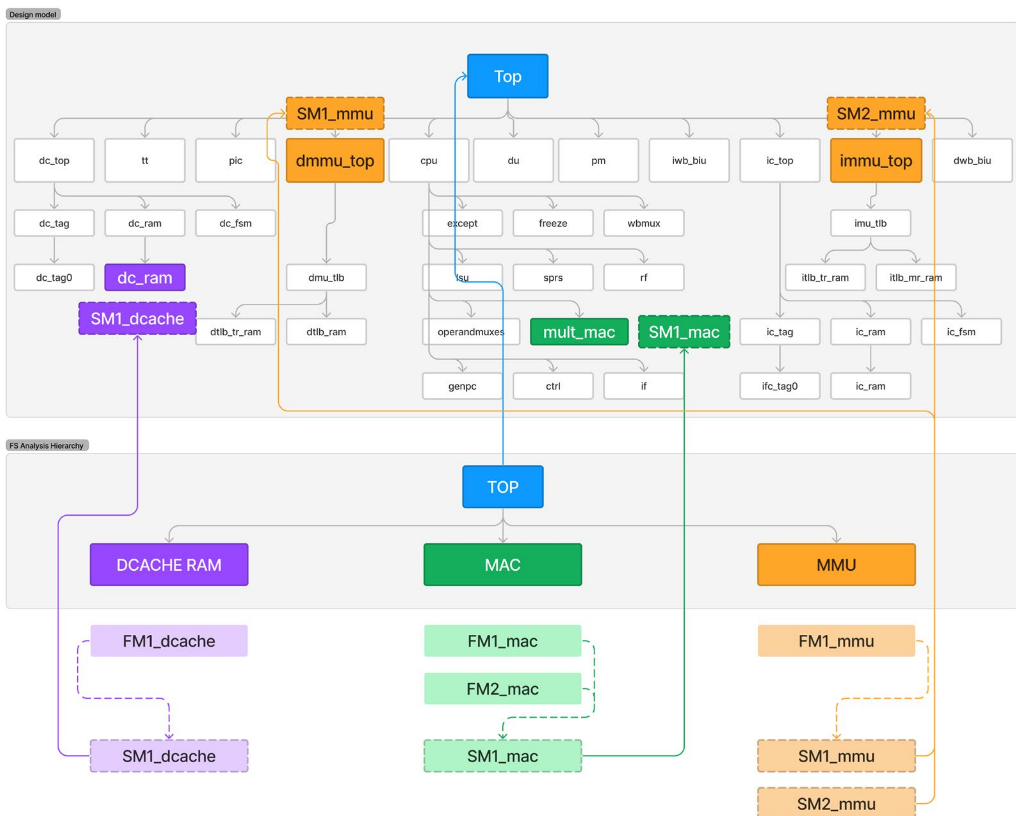


Figure 10. Mapping of the safety mechanisms to failure modes.

Technology Element Mapping

Technology elements represent the data model safety objects used to store the reliability information for a given design technology (e.g., digital, or standard cells, memories, etc.). By mapping technology elements to the failure modes, it is possible to compute the related failure rates. In the proposed data model, three basic modes are supported:

- Assumption-based FMEDA: The technology element has both the reliability information for the technology and the design information to be mapped to a given failure mode.
- Calculation-based FMEDA: The technology element has only reliability information and the design information is provided by the design mapping.

In Figure 11 two technologies elements are created: one for a memory technology and one for a digital technology. The two technology elements are mapped to the failure modes, linking the design information with reliability data, and enabling the failure rates computation.

The proposed data model supports mapping different technology elements to the same failure mode.

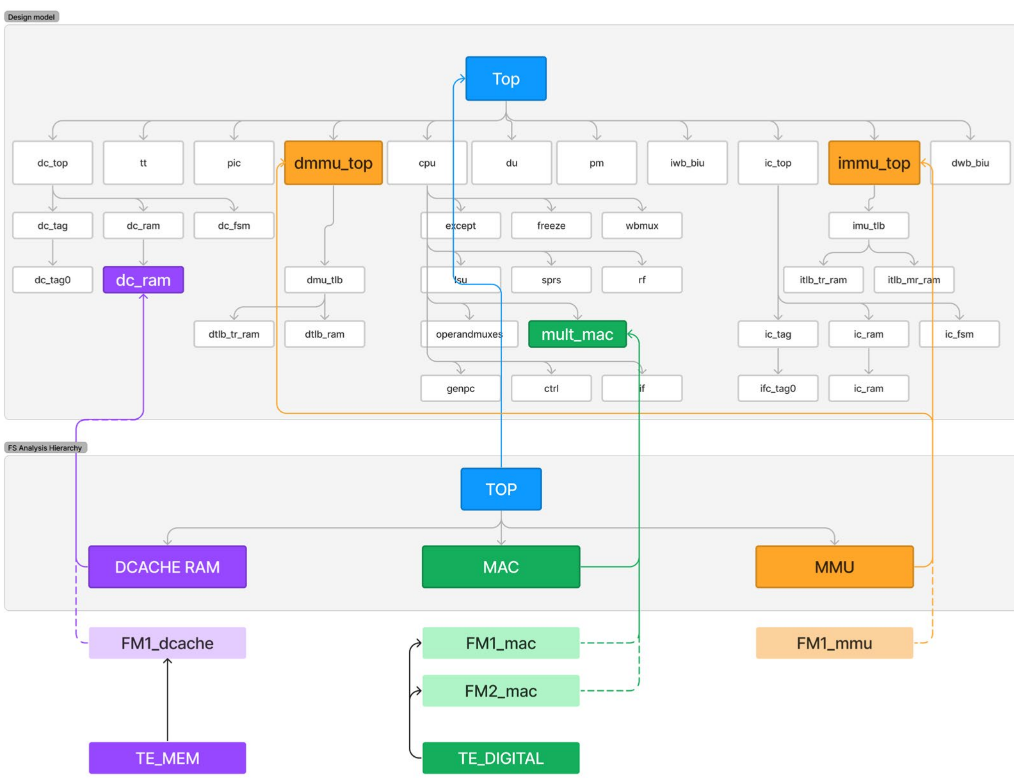


Figure 11. Mapping of the technology elements to failure modes.

Failure Mode Effects Mapping

The concept of Failure Mode Effect (FME) can be used to group, abstract, and finally "transport" failure rate contribution of one or more failure modes of a given safety analysis to a higher safety analysis scope. For example, it is possible to associate to an FME the weighted contribution of failure modes from an IP FMEDA in order to have a specific and desired safety metrics reporting at the SoC level.

If the end user wants to keep consistency of the FME reporting at different abstraction levels (e.g., IP vs SoC), constraints could be implemented on top of the data model, for example between the applied weights (e.g., sum of the weights for a given FME for all failure modes in the analysis to be 100%).

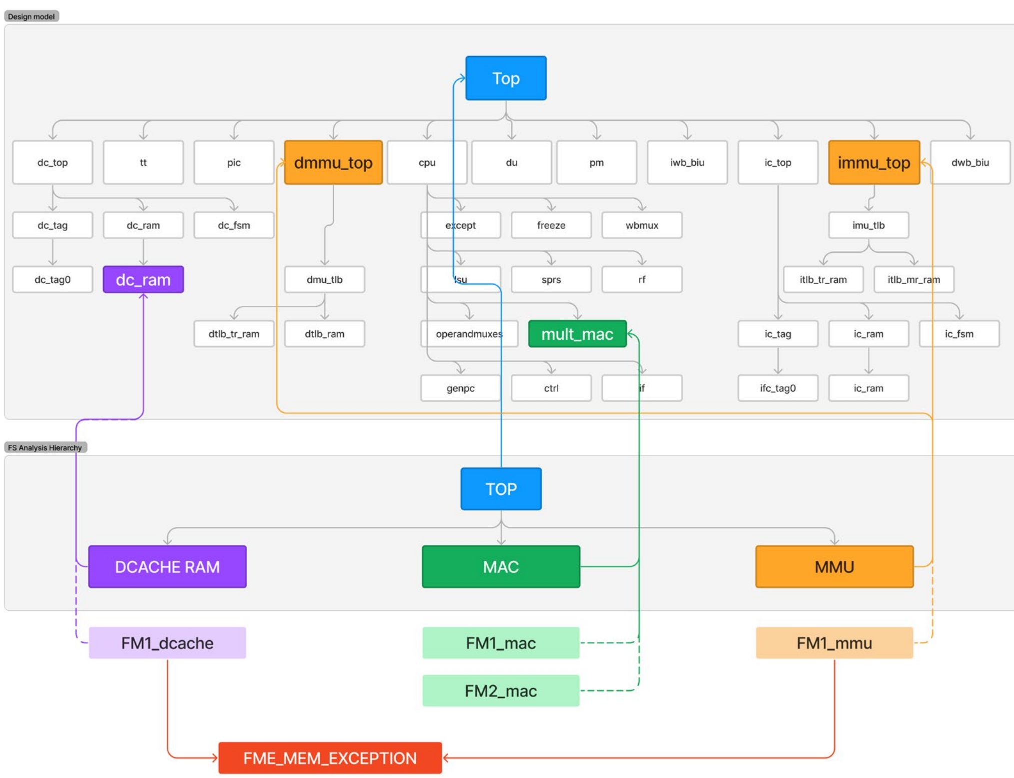


Figure 12. Creation of a Failure Mode Effect.

Complex Use Cases

The example provided during this section highlights a simplistic but common implementation including:

- Definition of the FS hierarchy
- Failure modes for each element of the hierarchy
- Safety mechanisms protecting those failure modes
- Technology elements within the failure modes
- Failure mode effects

The data model supports the broad range of FMEDA permutations such as FMEDAs with two or more parts and FMEDAs requiring greater than two levels of FS hierarchy depth.

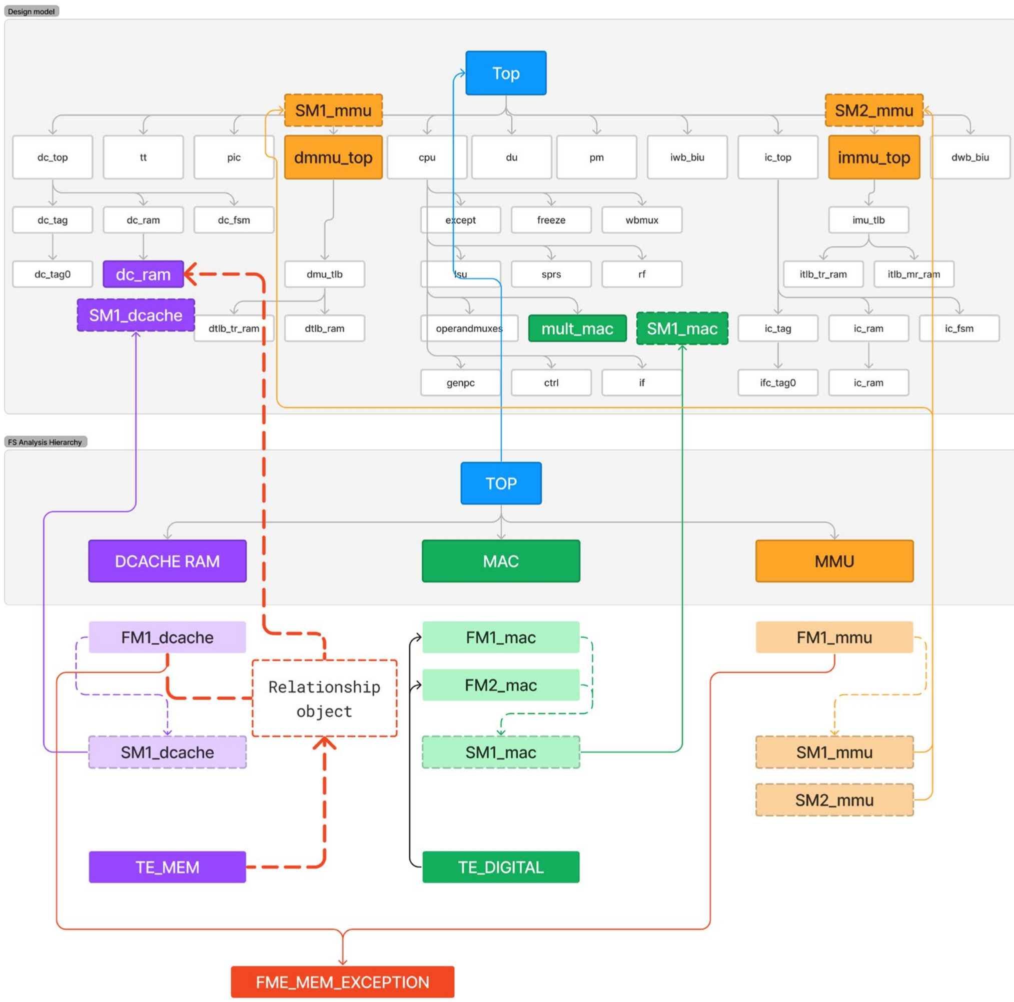


Figure 13. Final structure of the example.

The left-hand side branch of the diagram above shows a data model-accurate representation of mapping to design hierarchy operations. Mapping to design happens through the dashed red "Relationship object" box that connects Technology element **TE_MEM**, Failure mode **FM1_dcache**, and design hierarchy **Top.dc_top.dc_ram.dc_ram**. A detailed description of a relationship object is given in Mapping Technology Element – Failure Mode.

IV. FMEDA Type

The Accellera Functional Safety Working Group has defined two types of FMEDA supported by the data model: Assumption-based and Calculation-based. The following sections describe features and differences.

A. Assumption-based

An assumption-based FMEDA relies on user estimations to compute failure rates and metrics. The FS analysis and failure mode hierarchies do not have a correlation or mapping to any real design hierarchy, and therefore the metrics are only estimated.

B. Calculation-based

A calculation-based FMEDA leverages design mapping to enable automated computation of failure rates and metrics. A calculation-based FMEDA has associations (mapping) to a real design hierarchy or “design representation,” allowing for quantitative analysis of FMEDA metrics. The total areas and the related failure mode distribution (FMD) by default are not manually assigned by the user but are derived from the design hierarchy.

C. Mixing FMEDA Types

It is important to note that the two FMEDA types can co-exist on the same DUA for different portions of the design. For example, by mapping a technology element, it is always possible to manually specify the failure mode area information (e.g., FM_Size_Permanent, FM_Size_Transient). This feature has a potential priority conflict with the real design information mapped to failure modes in a calculation-based FMEDA (e.g., FM_mapping). The data model attribute used to discriminate between an Assumption-based and Calculation-based FMEDA can be used to define the priority in case of a mixed-mapping scenario. For example, in an assumption-based FMEDA, manually provided design information will take precedence (see Figure 14). Alternatively, information coming from design mapping will take precedence in a calculation-based FMEDA (see Figure 15).

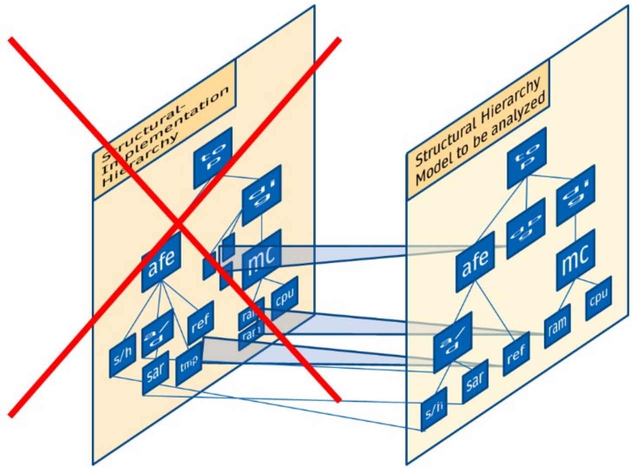


Figure 14. Assumption-based FMEDA.

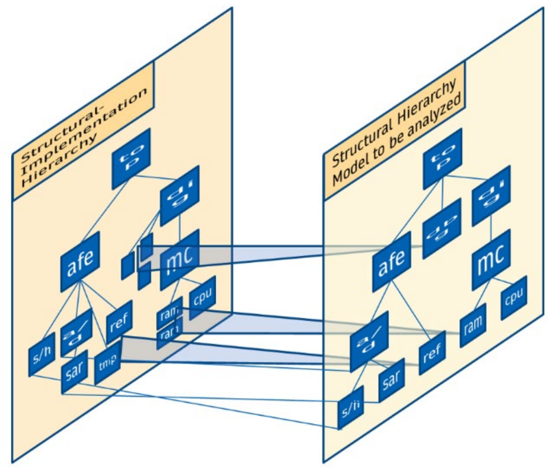


Figure 15. Calculation-based FMEDA.

Assumption-based FMEDA

- Defines the FS analysis hierarchy and FM hierarchy
- Does not have correlation/mapping to a design hierarchy and hence does not link to Design Objects
- Metrics are estimated and not derived from design objects

Calculation-based FMEDA

- FS analysis hierarchy is mapped/associated with the design hierarchy to quantify the failure rates and the failure mode distribution
- Failure modes can be mapped to their root cause, i.e., the portion of the design hierarchy that can trigger that failure mode
- Maps/Associates the FS analysis hierarchy and FM hierarchy to the design hierarchy (creates connections to predefined objects) to quantify FIT and FMD
- Includes Design Object identifiers of the design hierarchy associated to the textual description of the FS analysis hierarchy

V. Conceptual Data Model

A. Introduction to the Entity-Relationship Model

After the formalization of the FMEDA process, we extract a conceptual data model to represent the data needed to perform an FMEDA and exchange an FMEDA report. The goal of the conceptual data model is:

- To define and detail the information content (FS data) needed to perform the Functional Safety activities and generate the work products
- NOT to provide a reference implementation
- To be a systematic approach to define a language/format

A conceptual data model [12] [11]:

- Defines WHAT the system contains
- Does NOT define HOW the system should be implemented

In this work, to capture the conceptual data model, we rely on the well-known entity-relationship model [13] [14], and this section gives a brief summary of some if its terminology.

The three basic tenants of the Entity Relationship model are:

- **Entity**: A real-world thing
- **Attribute**: Characteristics or properties of an entity
- **Relationship**: Dependency or association between two entities

In addition to the three objects above, we will also rely on the concept of **Weak Entity**. While Entities are uniquely identified by a primary key, a weak entity is an entity that cannot be uniquely identified by its attributes alone; therefore, it must use a foreign key in conjunction with its attributes to create a primary key. A simple example is an employer that has a database of its employees, each represented as an entity with their unique employee ID. In this case, each employee can have one (or more) dependents. The dependents are weak entities because they do not have a unique ID by themselves, but only exist in the context of the employee (and their unique ID).

B. General Considerations

Key points about the conceptual data model:

- The data model is in addition to the existing design standards (see Figure 11 from [1]).
- We use an entity-relationship data model to capture the content.
- The high-level categories (i.e., entities) are derived from the FMEDA process defined in (reference to previous section) to meet the use cases.
- The detailed data model is represented in a table form to keep it generic.

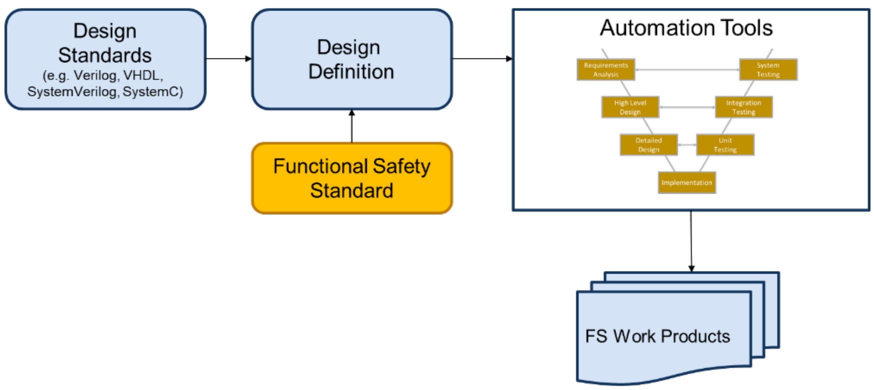


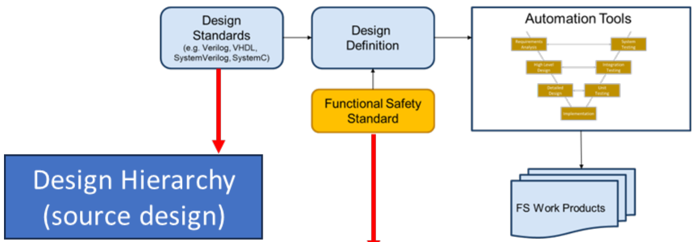
Figure 16. Functional Safety Standard in addition to the standard design representations.

Several options have been considered and discussed during the definition of the data model. The rationale used for the data model is to start with the simplest model and add complexity only if there are specific cases that are not supported. In fact, in general the more complex/flexible the model, the more rules are then needed to ensure consistent/exchangeable models. Even though the data model has been derived to support the FMEDA for a single IP/device, the validation process covers the hierarchical combination of multiple FMEDAs as a use case.

The high-level data categories are identified directly one-to-one from the data objects in the FMEDA process formalization described in Figure 7. In other words, the data model represents the implementation of the requirements defined as the formal FMEDA process.

Figure 17 reports the description of the high-level categories identified for the FMEDA process and the corresponding entity of the data model in which they are captured, connecting them to the role they play in the design definition captured in Figure 16. Each entity will then include several attributes to describe its properties, as detailed in Annex A – Data Model.

Objects in the table included in Figure 17 are also indicated with the same colored symbol used in Figure 7 to highlight the direct traceability existing between the process formalization and the definition of the conceptual data model. The data model has been defined as a direct traceable derivation of the data and data mapping used in the FMEDA process. The complete design definition and the scope in which the objects are defined is represented in a hierarchical way in Figure 18.



Functional Safety Data Model

FMEDA process data	Entity Type	Information Type	
FMEDA	FMEDA	Object	Object
FS Analysis Hierarchy	Element	Object	Object
FM Hierarchy	Failure_Mode	Object	Object
Technology Element	Technology_Element	Object	Object
Safety Mechanism Library	Safety_Mechanism	Object	Object
FM Effects	Failure_Mode_Effect	Object	Object
SM Mapping	SM-FM	Relationship	Relationship
FM Effects Mapping	FM-FME	Relationship	Relationship
Technology Element Mapping	TE-FM	Relationship	Relationship
Technology Element Mapping	TE-Element	Relationship	Relationship
Design Mapping	Inside the TE-FM since there is no Design Hierarchy in the data model	Relationship	Relationship
Design Mapping	Inside the TE-Element since there is no Design Hierarchy in the data model	Relationship	Relationship
Calculated FR	FR_ISO26262	Weak object (*)	Weak object
Calculated metrics	Metrics_ISO26262	Weak object (*)	Weak object
Calculated FR	FR_IEC61508	Weak object (*)	Weak object
Calculated metrics	Metrics_IEC61508	Weak object (*)	Weak object

Figure 17. Information included in the Functional Safety data model, derived from the process in Figure 7.

The following points describe the rationale applied for the attribute definition:

- Allow flexibility to support use cases without sacrificing ease of use.
- Select attributes to allow the smallest granularity needed.
- In specific cases, allow attributes for convenience only if they support significant ease of use.
- If the same attribute is defined on different entities, also specify the rules to reconcile the values in case of discrepancy/inconsistency if a single value is used for metrics calculation.
- An attribute is defined as required if the parsing of the data model will fail if that attribute is not provided.
- Other secondary criteria are readability and compactness of the model.

Detailed description of the data model and a derived language can be found in Annex A – Data Model and Annex B – Language, respectively.

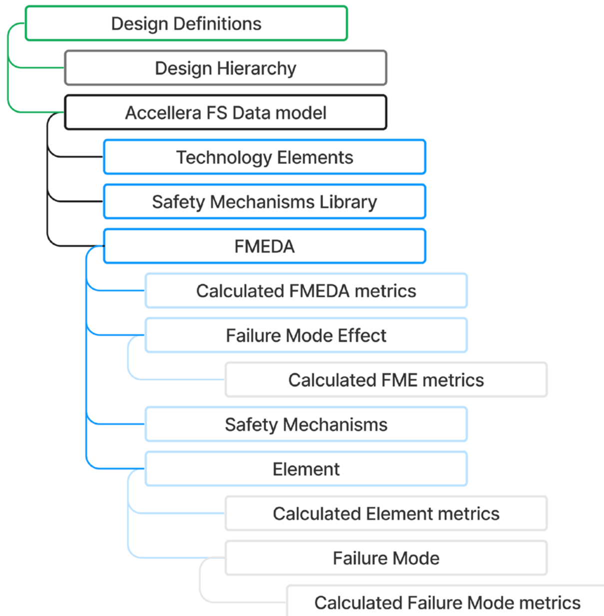


Figure 18. Design Definition and scope of the objects.

VI. Detailed Annotations on the Data Model

This section captures some of the methodology discussions that emerged as part of the data model definition; these discussions support the choices made for the content included in the data model itself.

A. FMEDA Type (Assumption-based, Calculation-base)

The FMEDA_type attribute defines the source of failure mode distribution data in case a choice needs to be made.

The failure mode distributions can be calculated based on:

- Estimations provided with the options fm_size or element_size
- Design metrics extracted from the design mapping as specified in the fm_mapping and element_mapping

When both options (*_size and *_mapping) are specified for an FM or element, the FMEDA type will select as follows:

- Assumption-based: The *_size takes precedence over *_mapping.
- Calculation-based: The *_mapping takes precedence over *_size.

This choice reflects the following intent:

- Assumption-based: The size of the FM or element is provided by the user.
- Calculation-based: The size of the FM or element is extracted by the mapping to the design hierarchy and the corresponding design metrics.

Even though the parameter is defined at the FMEDA level, the granularity of the choice can be applied to each individual FM or Element by the usage of the *_size and *_mapping parameters.

An example of usage is to start with an assumption-based FMEDA and then switch the attribute FMEDA_type to calculation-based and continue with design metrics extracted from the design representation. When the FMEDA_type is calculation-based, a design representation (see Figure 16) is also expected to be provided along with the FS data model.

B. FS Hierarchy and FM Hierarchy

The FS hierarchy (i.e., the Element objects) and FM hierarchy are defined in the context of an FMEDA.

The FMEDA, element, and FM have a required attribute “Name” that only needs to be unique in the context of the parent.

The FM also has an optional ID attribute that is instead unique inside an FMEDA.

An example of valid usage is the following:

FMEDA_Name	Part_Name	Subpart_Name	FailureMode_Name	FailureMode_ID
FMEDA_top_1	CPU_1	ALU	Wrong Data Computation	1
			Incorrect Adder Output	2
		Register File	Incorrect Data Value in the Register File	3
			Wrong Data Computation	4
	CPU_2	ALU	Wrong Data Computation	5
			Incorrect Adder Output	6
		Register File	Incorrect Data Value in the Register File	7
			Wrong Data Computation	8
FMEDA_top_2	CPU_1	ALU	Wrong Data Computation	1
			Incorrect Adder Output	2
		Register File	Incorrect Data Value in the Register File	3
			Wrong Data Computation	4
	CPU_2	ALU	Wrong Data Computation	5
			Incorrect Adder Output	6
		Register File	Incorrect Data Value in the Register File	7
			Wrong Data Computation	8

Or more generically:

FMEDA_Name	Part_Name	Subpart_Name	FailureMode_Name	FailureMode_ID
FMEDA_IP1	Part_1	Subpart_1	FailureMode_1	ID_1
			FailureMode_2	ID_2
		Subpart_2	FailureMode_2	ID_3
			FailureMode_3	ID_4
	Part_2	Subpart_1	FailureMode_1	ID_5
			FailureMode_2	ID_6
		Subpart_2	FailureMode_2	ID_7
			FailureMode_3	ID_8
FMEDA_IP2	Part_1	Subpart_1	FailureMode_1	ID_1
			FailureMode_2	ID_2
		Subpart_2	FailureMode_2	ID_3
			FailureMode_3	ID_4
	Part_2	Subpart_1	FailureMode_1	ID_5
			FailureMode_2	ID_6
		Subpart_2	FailureMode_2	ID_7
			FailureMode_3	ID_8

C. Technology Element

The term technology is commonly referred to as a technology node (e.g., 16nm, 7nm). The term technology element refers here to different elements of a technology node, such as RAM, digital, analog, and so on.

The Base Failure Rate (BFR) is the Failure rate of a unit design element. For digital technology elements, the unit design element is often defined as the smallest nand2 cell or as a transistor. For memories, the unit design element is the bit, while for analog it is the transistor. The BFR can be calculated as the FR of the whole design (λ_die) normalized to the unit design element. The BFR is provided as attributes *FR_Permanent*, *FR_Transient* and *FR_Transient_Derating* in the Technology entity.

Digital

For permanent faults, the Failure Rate (Raw Fit) for the Failure Mode (FM) is then calculated starting from the FR_perm as:

$$\lambda_{FM} = FR_{perm} \times \#FM_{unit\_design\_elements}$$

Or equivalently:

$$\lambda_{FM} = FR_{perm} \times \frac{FM_{size\_permanent}}{unit\_design\_element\_size}$$

For permanent faults, $FM_{size\_permanent}$ is assumed to have the whole area contribution of combinatorial and sequential logic.

For transient faults, the Failure Rate (Raw Fit) for the Failure Mode (FM) is then calculated starting from the FR_trans as:

$$\lambda_{FM} = FR_{tran} \times \frac{FM_{size\_tran}}{unit\_design\_element\_size} + FR_{tran\_comb} \times \frac{FM_{size\_perm} - FM_{size\_tran}}{unit\_design\_element\_size}$$

In case the FR_{tran_comb} is not available, it is possible to assume that the contribution of the combinatorial logic to the λ_{FM} can be obtained as a percentage of the failure rate transient:

$$FR_{tran\_comb} = FR_{trans} \times FR_{trans\_derating}$$

RAM/ROM/Flash

For permanent and transient faults, the Failure Rate (Raw Fit) for the Failure Mode (FM) is then calculated starting from the BFR as:

$$\lambda_{FM} = FR \times FM_{size\_bits}$$

Where FR is FR_{perm} and FR_{trans} respectively.

Analog

The permanent Failure Rate can be calculated as:

$$\lambda_{FM} = FR_{perm} \times \#FM_{unit\_design\_elements} = \frac{FM_{size\_permanent}}{unit\_design\_element\_size}$$

Note that the Failure Rate derating due to degradation of BFR characteristics based on the expected environmental conditions of the application is not accounted for in these formulas, but rather assumed to be included as part of the BFR.

D. FS Hierarchy Modeling

In the final reporting of ISO2626 [2], several levels of hierarchy can be compressed into a single subpart. However, to allow for flexibility to handle the complexity of modern SoCs, the data model also allows several levels of subparts in the FS hierarchy. The final reporting can still be compressed to have a single level of parts and subparts.

The following properties apply to the data model:

- Only one level of Part is supported in the FS Hierarchy.
- Several levels of Subparts are supported in the FS Hierarchy.
- A leaf is defined as an Element (part or subpart) with no children subparts (or, equivalently, only FM children).
- FM can be defined only on leaves of the FS hierarchy.
- An element of type Subpart can only have an element of type Part as a parent (in other words, the FS hierarchy cannot be FMEDA → Subpart).
- For calculation-based FMEDA, the design hierarchy mapped to parts and subparts cannot overlap.

Figure 19 below shows a few examples of FS hierarchy definitions that are allowed or not allowed:

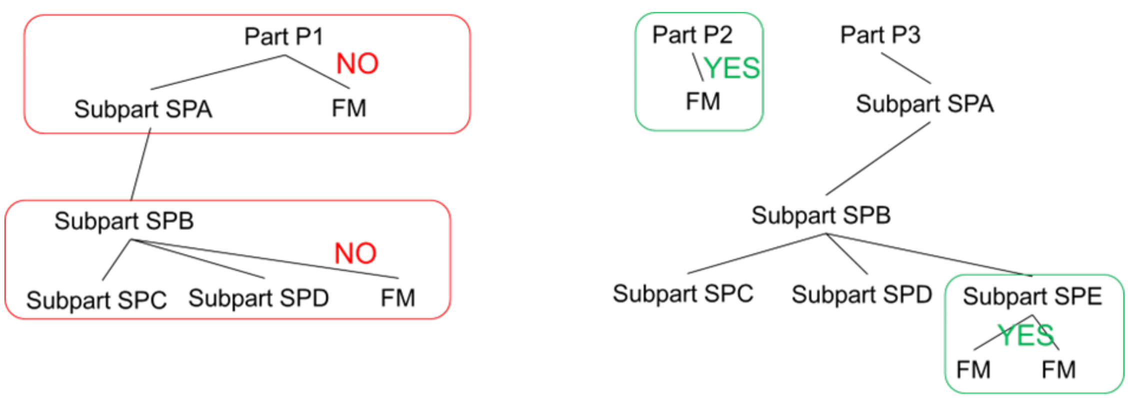


Figure 19. Examples of functional safety hierarchy definitions.

In ISO26262, the concept of Elementary SubPart (ESP) is also present and is defined as the leaf level of the FS hierarchy: "…smallest portion of a hardware subpart (3.73) considered in safety (3.132) analysis."

In practice, however, ESP is often used to partition the design hierarchy into a finer granularity, often implementing a specific methodology based on cone-of-logic extraction, and therefore describes parts of a design hierarchy instead. In other words, ESPs are used in practice to gather portions of HW logic and build them into an FM root cause. Other examples referred to as ESP in ISO can be modeled with the mapping attribute.

Therefore, we will not introduce the concept of ESP in the data model, but we will consider instead whether we need to define a specific operation/support for design hierarchy manipulation/aggregation/partitioning in case a finer granularity is needed. The addition of this support would have the goal to capture the design manipulation/aggregation/partitioning in an implicit rather than explicit way and avoid potentially excessive data transfer.

E. Operations on Design Mapping

Logic operations on design mapping can be convenient and allow for an implicit description rather than an explicit, potentially large, list of design elements. The most useful operation is subtraction or exclusion. The parameters Part_Mapping_Exclude, SubPart_Mapping_exclude, and FM_Mapping_Exclude provide the capability; they define which design elements to exclude from the list of design elements mapped to a safety object. An example of usage is to provide a convenient way to exclude pervasive logic like BIST.

F. DC Aggregation Methods

Some FMs are covered by a single SM, and some FMs are protected by a combination of SMs. Both use cases are possible. When multiple SMs are covering the same FM, the overall resulting DC is the aggregation of the individual DC based on some heuristics/criteria. The data model includes an FM attribute called DC_aggregation that supports the definition of several heuristics:

- max: The max DC of all SMs is selected
- sum: The DC from all SM are summed up and capped to 100%
- residual: The DC is calculated with the following formula:

$$DC_{residual} = 100\% - \prod(100\% - DC_{SM})$$

The DC_aggregation attribute is defined separately for Transient and Permanent.

An example is included in Table 3, where a single FM is covered by several SMs called SM1, SM2, and SM3, and the value of the DC associated with each SM is defined. The last three columns show the resulting DC obtained using each of the heuristic.

Table 3. Example of a single FM covered by three SMs: SM1, SM2, and SM3

FM	DC-SM1	DC-SM2	DC-SM3	Max	Sum	Residual
FM1	30%	60%	90%	90%	100%	97.2%

Specifically, and for sake of example, these are the detailed calculations for the $DC_{residual}$:

1. $100\% - [(100\% - 30\%) \times (100\% - 60\%) \times (100\% - 90\%)]$
2. $100\% - [70\% \times 40\% \times 10\%]$
3. $100\% - 2.8\% = 97.2\%$

G. Failure Mode Effect

A Failure Mode Effect (FME) represents the consequence of a failure mode seen at the top level of the DUA when a fault occurs in the DUA. For a Safety Element out of Context (SEooC), the FME is based on assumptions of how the DUA is going to be used at the next layer. The receiving layer needs to validate the assumptions and map the IP FME to the component FM (and similarly for the other layers of the supply chain). In other words, the FME captures the information of the interface between various levels/layers of the FMEDA analysis.

The relationship between FM and FME can be many-to-many:

- An FM can contribute to multiple FMEs.
- An FME can be caused by many FMs.

Also, there are no limitations or rules on how to map FMs to FMEs in terms of where the FM belongs in the FS hierarchy.

The contribution of FMs to FMEs can be assigned a weight as well. For a given FM, the sum of its contributions to the FME adds up to 100%. The assumption behind this is that the FMEs are non-overlapping. Table 4 shows a case of ill-defined FMEs where this assumption is not fulfilled. In this case, the sum of the FMEs is beyond 100% because FME_B and FME_C are overlapping., i.e., a packet could be "corrupted" (FME_B) and "at a wrong time" (FME_C) at the same time. A better way to define FMEs is in Table 5.

Defined in this way, the total residual FR for the device (due to all of the FMs) is maintained and distributed in a different view across the FMEs.

Table 6 provides an example of the FS hierarchy, the FM hierarchy, and the FMEs for IP1 (the DUA) and shows the information about the FM to FME mapping, including the corresponding weights. Figure 20 reports a graphical view of part of the same example, where the FMs of IP1/P1/SP1 and IP1/P2/SP2 are mapped to the FMEs (FME_A, FME_B, FME_C, FME_D, and FME_E).

Table 4. Example of ill-defined, overlapping FMEs.

FME_A	Packet not generated when it should be	30%
FME_B	Packet corrupted	35%
FME_C	Packet at wrong time	35%

Table 5. Example of well-defined, no overlapping FMEs.

FME_A	Packet not generated when it should be	30%
FME_B	Packet corrupted but at right time	20%
FME_C	Packet at wrong time	20%
FME_C2	Packet at wrong time and corrupted	30%

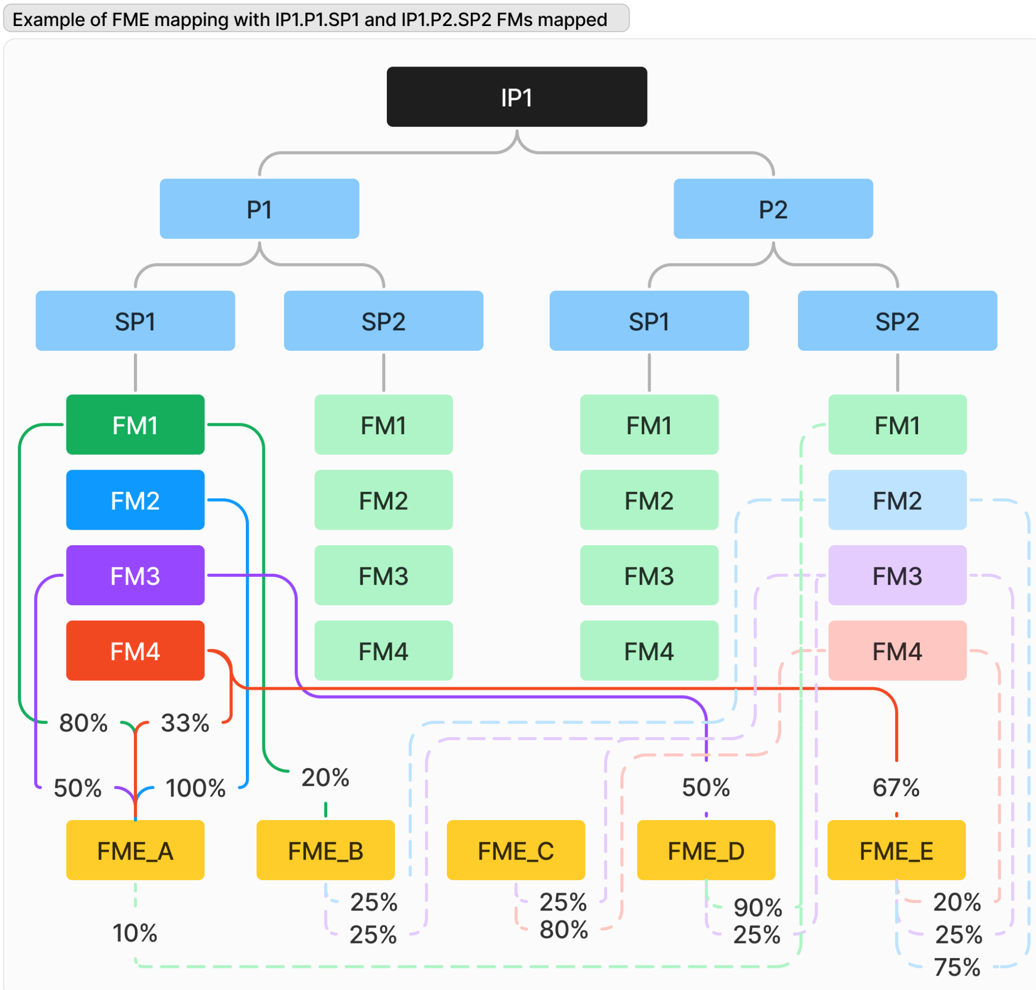


Figure 20. Example of FME mapping with FMs of two subparts mapped.

Table 6. Example of FME mapping with all FMs mapped to FMEs.

Part	Subpart	Failure Mode	IP1: FME_A	IP1: FME_C	IP1: FME_D	IP1: FME_E
P1	SP1	P1_SP1_FM1	80%	20%		
		P1_SP1_FM2	100%			
		P1_SP1_FM3	50%		50%	
		P1_SP1_FM4	33%			67%
	SP2	P1_SP2_FM1		59%	41%	

		P1_SP2_FM2		50%	50%	
		P1_SP2_FM3		10%		90%
		P1_SP2_FM4		20%	80%	
P2	SP1	P2_SP1_FM1	100%			
		P2_SP1_FM2	100%			
		P2_SP1_FM3	100%			
		P2_SP1_FM4	100%			
	SP2	P2_SP2_FM1	10%		90%	
		P2_SP2_FM2	25%		75%	
		P2_SP2_FM3	50%		50%	
		P2_SP2_FM4				100%

There is no prescription of how the IP1 FMEDA is then abstracted to be provided from the provider to the consumer/integrator. Any combination is possible and independent of the data model definition. Examples of use cases are included in Table 7 and Table 8: Table 7 shows maximum compression, which is typically used for non-configurable IPs, while Table 8 maintains higher granularity keeping the subparts of the original FMEDA and it is typically used for configurable IPs.

Since FMEDA analysis uses a bottom-up, the definition of FME and their connection is also defined bottom-up. Based on the intended functionality, the user defines the FMEs for the device and then maps the FMs to the FMEs. The specific weight of a FM on the FME can also be assigned.

Following the example above, the user would follow this process:

1. Define the FMEs: Create IP1:FME_A, IP1:FME_B, IP1:FME_C, IP1:FME_D, IP1:FME_E
2. Map each FM to all of the FMEs to which it contributes:
   a. Map P1_SP1_FM1 to IP1:FME_A with weight 80%
   b. Map P1_SP1_FM1 to IP1:FME_B with weight 20%
   c. Map P1_SP1_FM2 to IP1:FME_A with weight 100%
   d. …

Table 7. Example of FMEDA abstraction doing high compression.

Part	Subpart	Failure Mode
IP1	IP1	IP1:FME_A
		IP1:FME_B
		IP1:FME_C
		IP1:FME_D
		IP1:FME_E

Table 8. Example of FMEDA abstraction maintaining higher granularity.

Part	Subpart	Failure Mode
P1	SP1	IP1:FME_A
		IP1:FME_B
		IP1:FME_C
	SP1	IP1:FME_D
		IP1:FME_E
P2	SP1	IP1:FME_A
	SP2	IP1:FME_A
		IP1:FME_D
		IP1:FME_E

Table 9, Table 10, and Table 11 cover an example of how the IP1 FMEDA would be summarized and then integrated at a component level. Table 9 is the IP1 FMEDA, Table 10 is the IP1 summary FMEDA and Table 11 is the component FMEDA in which IP1 is instantiated, together with IP2, IP3 and so on.

Table 9. Example of IP1 FMEDA.

Part	Subpart	Failure Mode	IP1: FME_A	IP1: FME_B	IP1: FME_C	IP1: FME_D	IP1: FME_E
P1	SP1	P1_SP1_FM1	x	x			
		P1_SP1_FM2	x				
		P1_SP1_FM3	x			x	
		P1_SP1_FM4	x				x
	SP2	P1_SP2_FM1		x	x		
		P1_SP2_FM2		x		x	

		P1_SP2_FM3		x			x
		P1_SP2_FM4	x	x	x	x	
P2	SP1	P2_SP1_FM1				x	
		P2_SP1_FM2			x		
		P2_SP1_FM3			x	x	
		P2_SP1_FM4		x	x		
	SP2	P2_SP2_FM1	x				
		P2_SP2_FM2		x		x	
		P2_SP2_FM3	x	x	x		
		P2_SP2_FM4			x	x	

Table 10. Example of summarized IP1 FMEDA.

Part	Subpart	Failure Mode
IP1	IP1	IP1: FME_A
		IP1: FME_B
		IP1: FME_C
		IP1: FME_D
		IP1: FME_E

Table 11. Example of IP1 FMEDA summarized and instantiated.

Part	Subpart	Failure Mode
IP1	IP1	IP1: FME_A
		IP1: FME_B
		IP1: FME_C
		IP1: FME_D
		IP1: FME_E
IP2	IP2	…
…		
IP3	IP4	…
…		

VII. Concluding Remarks

The first Accellera FS WG White Paper [1] details the goals and scope of the work covered in the working group. This Accellera FS WG Data Model White Paper focuses instead on defining the FS content necessary to create, modify, and exchange an FMEDA. The work follows a three-step process: formalize the FMEDA process, extract from it a data model, and then define the corresponding language to capture the data model itself. This paper contains the first two steps, and the details of the data model are illustrated using a sample language, which can be subject to change. The actual language constitutes the Accellera FS standard and will be finalized and formalized in the LRM. The goal of the Accellera FS standard is to facilitate the implementation of best practices defined in existing safety standards, such as ISO26262 [2] and IEC61508 [3].

H. Accellera FS WG Supporting Entities

We are thankful to our supporting entities (in alphabetical order): Agnisys, Inc., Allegro MicroSystems, AMD, ams Osram, Analog Devices, ARM Ltd., Arteris, Inc., Breker Verification Systems, Inc., Cadence Design Systems, Inc., COSEDA Technologies GmbH, Doulos Ltd., Fraunhofer Institute For Integrated Circuits, Huawei Technologies Sweden AB, Infineon Technologies AG, Intel Corporation, Marvell International Ltd, NVIDIA Corporation, NXP Semiconductor, Perforce, Qualcomm, Renesas Electronics Corp., Robert Bosch GmbH, Shanghai UniVista Industrial Software Group, Siemens EDA, SiFive, STMicroelectronics, Synopsys, Technical University Dortmund, Texas Instruments, Vayavya Labs, XEPIC Corporation.

I. Acknowledgements

Special recognition (in alphabetical order by last name) for their contributions to the Accellera FS WG and Proposed WG with discussions, brainstorming, examples, and writing/reviewing of this white paper: Jyotika Athavale, Oscar Ballan, Pramod Bhardwaj, Alexis Boutillier, Samir Camdzic, Jason Campbell, Giuseppe Capodanno, Bala Chavali, Shivakumar Chonnad, Teo Cupaiuolo, Kaushik De, Yakov Felikman, Darren Galpin, Franck Galtie, Joerg Grosse, Regis Gubian, Mark Hampton, John Hayden, Ghani Kanawati, Francesco Lertora, Thiyagu Loganathan, Stefano Lorenzini, Nir Maor, Shrenik Mehta, Alessandra Nardi, Meirav Nitzan, Alexandre Palus, Arpita Potdar, Vatsa Prahallada, Om Ranjan, Kevin Rich, Rolf Schlagenhaft, Francesco Sforza, Ivano Shivananda Troja, Ashish Vanjari, Federico Venini, Riccardo Vincelli, Prasanth Viswanathan Pillai, Evgeny Vlasov, Ekaterina Vlasova, Jens Warmuth, Jacob Wiltgen.

VIII. Annex A – Data Model

This chapter describes data model objects, their attributes, and additional properties of attributes: name, type, description, and whether an attribute is required. The assumption is that if the attribute is required, the parsing of the data model will fail if that attribute is not provided.

The full list of objects defined according to the data model v0.1 is as follows:

- FMEDA
- Element
- Failure Mode
- Technology Element
- Safety Mechanism
- Failure Mode Effect
- Mapping Safety Mechanism – Failure Mode
- Mapping Failure Mode – Failure Mode Effect
- Mapping Technology Element – Failure Mode
- Mapping Technology Element – Element
- Define ISO26262 Failure Rate
- Define ISO26262 Metric
- Define IEC61508 Failure Rate
- Define IEC61508 Metric

A full ERD is presented on Figure 21. It shows a high-level overview of available objects and their attributes, as well as the connections between them. For detailed information regarding each object, use links from above.

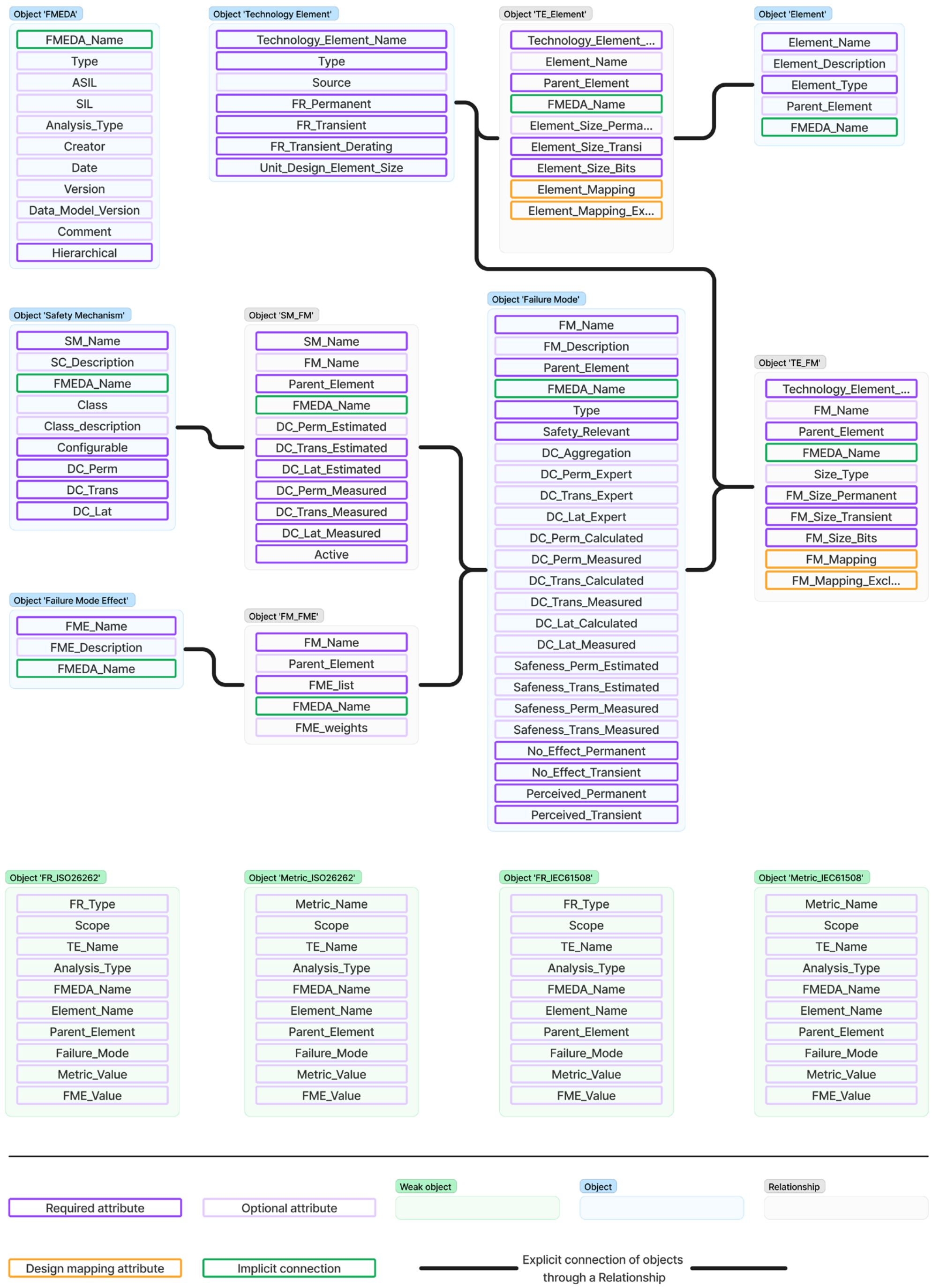


Figure 21. Entity Relationship Diagram of the data model.

Implicit hierarchical relationships of objects are not shown on the ERD. An example of such implicit connection is shown using FMEDA attributes that are highlighted in green.

1 A. FMEDA

Entity name FMEDA

Key identifier FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
FMEDA_Name	String	Name (identifier) of the FMEDA of the project.	Yes
Type	Enumerate {assumption-based, calculation-based}	Defines the source of the failure mode distribution in case a choice needs to be made. The failure mode distributions can be calculated based on: • Estimations provided with the options fm_size or element_size • Design metrics extracted from the design mapping as specified in the fm_mapping and element_mapping When both options (*_size and *_mapping) are specified for an FM, the FMEDA type will select as follows: • assumption-based: The *_size takes precedence over *_mapping • calculation-based: The *_mapping takes precedence over *_size	No
ASIL	Enumerate {None, A, B, C, D}	Defines the ASIL target for the FMEDA (for a given Safety Goal) according to ISO26262. Used also to specify that the FMEDA is for ISO26262.	No
SIL	Enumerate {None, 1, 2, 3, 4}	Defines the SIL target for the FMEDA according to IEC61508. Used also to specify that the FMEDA is for IEC61508.	No
Analysis_Type	List of Enumerate {Permanent, Transient, All}	Defines the failure types to be considered and which metrics to be calculated within the safety analysis. More than one value can be specified, e.g., Analysis_Type = {Permanent} or	Yes

		Analysis_Type = {Permanent, Transient} The value “All” implies all Failure Types are activated. Defined as “All” instead of “Both” allows for plans for more than just Transient and Permanent.	
Creator	String	Name of the company that generated the FMEDA.	No
Date	Date	Date when the FMEDA was generated.	No
Version	Float	Version of the FMEDA.	No
Data_Model_Version	Float	The version of the data model (not the FMEDA version).	No
Comment	String	Information that does not have a specific field in the FMEDA object.	No
Hierarchical	Enumerate {Yes, No}	Describes whether the FMEDA is fully flat or hierarchical, meant as aggregation of other FMEDAs	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values	No

2

3
B. Element

Entity name Element

Key identifier Element_Name + Parent_Element + FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
Element_Name	String	Name (identifier) of the Element.	Yes
Element_Description	String	Description of the intended functionality of the Element.	No
Element_Type	Enum {System, Element, SubElement, Component, SubComponent, Part, SubPart}	Specifies the type of the Element. Element_Type = Component or SubComponent can only be defined if the analysis is for IEC61508, inferred from the FMEDA entity, whether it has ASIL or SIL defined.	Yes
Parent_Element	String	Connects the Element to its Parent in the FS hierarchy.	No
FMEDA_Name	String	Connects the FS hierarchy to the FMEDA project.	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

4

5 C. Failure Mode

Entity name Failure Mode

Key identifier Failure_Mode_Name + Parent_Element + FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
FM_Name	String	Name (identifier) of the Failure Mode.	Yes
FM_Description	String	Description of the Failure Mode.	No
Parent_Element	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
FMEDA_Name	String	Connects the FS hierarchy to the FMEDA project.	Yes
Type	Enumerate {Mission, Passive, Active}	Describes if and how the FM can violate a safety goal. • Mission: Participates in a safety function Diagnostic is broken into: • Passive: Participates in a safety mechanism • Active: Participates in a safety mechanism that can violate a safety goal	Yes
Safety_Relevant	Boolean {yes, no}	Specifies if the failure mode is safety related. Safety_Relevant = no is equivalent to the "no part" according to IEC61508.	Yes
DC_Aggregation	Enumerate {Max, Sum, Residual, Expert}	Defines the heuristic/algorithm used to aggregate the DC of multiple SMs applied to the same FM. If DC_aggregation = expert, then the value is provided by the user using the attribute DC_expert.	Yes

DC_Perm_Expert	Float [0, 100]	Allows the user to specify the Permanent DC of the FM aggregated over the SMs associated with an FM. Only available if DC_Aggregation = expert.	No
DC_Trans_Expert	Float [0, 100]	Allows the user to specify the Transient DC of the FM aggregated over the SMs associated with an FM. Only available if DC_Aggregation = expert.	No
DC_Lat_Expert	Float [0, 100]	Allows the user to specify the DC latent of the FM aggregated over the SMs associated with an FM. Only available if DC_Aggregation = expert.	No
DC_Perm_Calculated	Float [0, 100]	Stores the results of the total Permanent DC in case several SMs are defined for the FM. The algorithm followed to aggregate the DC of the multiple SMs is defined with attribute DC_aggregation. The DC of the individual SM will be either defined in the SM entity (DC_Perm) or in the SM-FM entity (DC_Perm_Estimated).	No
DC_Perm_Measured	Float [0, 100]	Store the value of the Permanent DC coming from Fault Injection activities. When present, it will take precedence over the DC_Perm_Calculated in the metrics calculations.	No
DC_Trans_Calculated	Float [0, 100]	Stores the results of the total Transient DC in case several SMs are defined for the FM. The algorithm followed to aggregate the DC of the multiple SMs is defined with attribute DC_aggregation. The DC of the individual SM will be either defined in the SM entity (DC_Trans) or in the SM-FM entity (DC_Trans_Estimated).	No
DC_Trans_Measured	Float [0, 100]	Store the value of the Transient DC coming from Fault Injection activities. When present, it will take precedence over the DC_Perm_Calculated in the metrics calculations.	No
DC_Lat_Calculated	Float [0, 100]	Stores the results of the total Latent DC in case several SMs are defined for the FM. The algorithm followed to aggregate the DC of the multiple SMs is defined with attribute DC_aggregation. The DC of the individual SM will be either defined in the SM entity (DC_Lat) or in the SM-FM entity (DC_Lat_Estimated).	No
DC_Lat_Measured	Float [0, 100]	Store the value of the Latent DC coming from Fault Injection activities. When present, it will take precedence over the DC_Lat_Calculated in the metrics calculations.	No
Safeness_Perm_Estimated	Float [0, 100]	Percentage/Fraction of safeness for permanent faults (i.e., faults that do not contribute to the violation of a safety goal).	No

Safeness_Trans_Estimated	Float [0, 100]	Percentage/Fraction of safeness for transient faults (i.e., faults that do not contribute to the violation of a safety goal).	No
Safeness_Perm_Measured	Float [0, 100]	Percentage/Fraction of safeness for permanent faults, (i.e., faults that do not contribute to the violation of a safety goal) as a result of Fault Injection Activities or other techniques to measure.	No
Safeness_Trans_Measured	Float [0, 100]	Percentage/Fraction of safeness for transient faults, (i.e., faults that do not contribute to the violation of a safety goal) as a result of Fault Injection Activities or other techniques to measure.	No
No_Effect_Permanent	Float [0, 100]	No effect Permanent rate according to IEC 61508. This can only be used if the FMEDA has SIL target values (attribute of the FMEDA entity).	Yes
No_Effect_Transient	Float [0, 100]	No effect Transient rate according to IEC 61508. This can only be used if the FMEDA has SIL target values (attribute of the FMEDA entity).	Yes
Perceived_Permanent	Float [0, 100]	Specifies the fraction of multi-point faults that are not detected but are perceived. This is only for ISO26262 (i.e., if the FMEDA has the ASIL level defined).	Yes
Perceived_Transient	Float [0, 100]	Specifies the fraction of multi-point faults that are not detected but are perceived. This is only for ISO26262 (i.e., if the FMEDA has the ASIL level defined).	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

6

7 D. Technology Element

Entity name Technology Element

Key identifier Technology_Element_Name

Attribute Name	Attribute Type	Description	Req'd
Technology_Element_Name	String	Name (identifier) of the Technology_Element.	Yes
Type	Enumerate {Digital, RAM, ROM, Flash, Analog}	Type of t=Technology	Yes
Source	Enumerate {IEC_62380, SN_25900, IEC_61709, Expert}	Description of the source of BFR data (e.g., IEC TR 62380, testing, field returns…).	No
FR_Permanent	Float [0,N]	Base Failure Rate (BFR) for permanent faults.	Yes
FR_Transient	Float [0,N]	Base Failure Rate (BFR) for transient faults.	Yes
FR_Transient_Derating	Float [0,1]	Derating of the BFR for transient faults in digital and analog technology elements. Used to account for the contribution of combinatorial logic to the raw fit transient (as a percentage of the raw fit transient from sequential/memory elements). For details see Technology Element.	Yes
Unit_Design_Element_Size	Float [0,N]	Area of the unit design element of the technology element. Required if Technology_Element = Digital/Analog. Not utilized if Technology_Element = RAM/ROM/Flash. It is used to calculate the number of unit design elements in an FM and hence calculates the raw FIT for the FM. The following formula applies: `*#FM_unit_design_elements = FM_size_permanent/unit_design_element_size*`. It should be set to 1 if the *FM_size_permanent/FM_size_transient* is already expressed in number of unit design elements.	Yes

User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

8 E. Safety Mechanism

Entity name Safety mechanism

Key identifier Safety_Mechanism_Name

Attribute Name	Attribute Type	Description	Req'd
SM_Name	String	Name (identifier) of the Safety Mechanism.	Yes
SM_Description	String	Description of the SM.	No
FMEDA_Name	String	Connects the FS hierarchy to the FMEDA project.	No
Class	Enumerate {HW, SW, AoU, AoU-SW, AoU-HW, user-defined}	Method by which the safety mechanism is to be realized. Notes: 1) AoU is to capture when the SM is not part of the product (potentially raise a flag during FMEDA integration) 2) HW allows for further specification for downstream tools	No
Class_description	String	Description of the class. This is specially meant in the case in which the class is user-defined, but available for all classes.	No
Configurable	Boolean {yes, no}	Captures whether the SM can be turned on or off by the user/integrator. If configurable=yes, then the "SM-FM active" attribute can be used.	Yes
DC_Perm	Float [0, 100]	Diagnostic coverage of the SM in isolation for permanent faults.	Yes
DC_Trans	Float [0, 100]	Diagnostic coverage of the SM in isolation for transient faults.	Yes

DC_Lat	Float [0, 100]	Diagnostic coverage of the SM in isolation for latent faults. This attribute is only available when the ASIL target level is defined. Not available if only the SIL target is defined.	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

To apply a diagnostic coverage specific to an SM-FM pair, use the DC_ *type* attribute in the SM-FM category. When SM:DC_ *type* and SM-FM:DC_ *type* are specified, the SM-FM:DC_ *type* attribute takes precedence. See Mapping Safety Mechanism – Failure Mode for details.

F. Failure Mode Effect

Entity name Failure Mode Effect

Key identifier Failure_Mode_Effect_Name + FMEDA_name

Attribute Name	Attribute Type	Description	Req'd
FME_Name	String	Name (identifier) of the FME.	Yes
FME_Description	String	Description of the FME.	No
FMEDA_Name	String	Connects the FME to the FMEDA project.	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

G. Mapping Safety Mechanism – Failure Mode

Entity name SM_FM

Key identifier Assignment_Name + FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
SM_Name	String	Name (identifier) of the SM applied to the FM.	Yes
FM_Name	String	Name (identifier) of the FM covered by the SM.	Yes
Parent_Element	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
FMEDA_Name	String	Connects to the FMEDA project.	Yes
DC_Perm_Estimated	Float [0, 100]	Diagnostic coverage of the SM applied to the FM for permanent faults.	No
DC_Trans_Estimated	Float [0, 100]	Diagnostic coverage of the SM applied to the FM for transient faults.	No
DC_Lat_Estimated	Float [0, 100]	Diagnostic coverage of the SM applied to the FM for latent faults.	No

DC_Perm_Measured	Float [0, 100]	Diagnostic coverage of the SM applied to the FM for permanent faults as a result of Fault Injection Activities.	No
DC_Trans_Measured	Float [0, 100]	Diagnostic coverage of the SM applied to the FM for transient faults as a result of Fault Injection Activities.	No
DC_Lat_Measured	Float [0, 100]	Diagnostic coverage of the SM applied to the FM for latent faults as a result of Fault Injection Activities.	No
Active	Boolean {yes, no}	Specifies whether the SM is enabled for this FM. Only accessible if the SM_Configurable attribute=yes.	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

DC_ *type* value is specific to the SM-FM pair and takes precedence over the DC_ *type* of the SM category. If such value is not specified, then the value is taken from the DC_ *type* attribute of the SM category.

18 H. Mapping Failure Mode – Failure Mode Effect

Entity name FM_FME

Key identifier Assignment_Name + FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
FM_Name	String	Name (identifier) of the FM contributing to the FME.	Yes
Parent_Element	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
FME_list	List of strings	List of names (identifiers) of the FMEs caused by the FM. Connects the FM to the FME that represents the consequence seen at the top level (of the DUA scope).	Yes
FMEDA_Name	String	Connects to the FMEDA project.	Yes
FME_weights	List of floats	Weights of the contributions of the FM to the list of FMEs defined in FME_list.	Yes
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

19

20 I. Mapping Technology Element – Failure Mode

Entity name TE_FM

Key identifier Assignment_Name + FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
Technology_Element_Name	String	Defines a technology element in which the FM is implemented.	Yes
FM_Name	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
Parent_Element	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
FMEDA_Name	String	Connects to the FMEDA project.	Yes
Size_Type	Enumerate {Percentage, Absolute, Uniform_Distribution}	Defines whether the FM_Size will be: • Percentage: A percentage of the parent Element_Size • Absolute: An absolute value • Uniform_Distribution: A uniform distribution of the parent Element_Size	Yes
FM_Size_Permanent	Float [0,N]	Specifies the size of the FM to calculate the FMD (FM distribution) for permanent faults for the associated Technology Element. This attribute is given precedence for an assumption-based FMEDA. Otherwise, the FMD is calculated based on the area of the FM defined by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM. In the semiconductor world, this is the size of the logic for which permanent faults	Yes

		can occur (combinatorial and sequential logic gates).	
FM_Size_Transient	Float [0,N]	Specifies the size of the FM to calculate the FMD (FM distribution) for transient faults for the associated Technology Element. This attribute is given precedence for an assumption-based FMEDA. Otherwise, the FMD is calculated based on the area of the FM defined by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM. In the semiconductor world, this is the size of the logic for which transient faults can occur (sequential logic gates, e.g., includes Flip-Flops, Latches and Register Files).	Yes
FM_Size_Bits	Integer [0,N]	Specifies the size of the FM to calculate the FMD (FM distribution) for transient and permanent faults for the associated Technology Element. This attribute is given precedence for an assumption-based FMEDA. Otherwise, the FMD is calculated based on the area of the FM defined by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM. In the semiconductor world, this is the size of the memory logic for which transient and permanent faults can occur.	Yes
FM_Mapping	List of Strings	Connects to the DUA representation and identifies the portion of the design responsible for the Failure Mode. This attribute is given precedence for a calculation-based FMEDA. Detailed semantics are to be defined in the LRM.	No
FM_Mapping_Exclude	List of Strings	Connects to the DUA representation and identifies the portion of the design to be excluded from the FM_Mapping. Can only be used in conjunction with the FM_Mapping attribute. This attribute is only used for a calculation-based FMEDA.	No
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

22 J. Mapping Technology Element – Element

Entity name TE_Element

Key identifier Assignment_Name + FMEDA_Name

Attribute Name	Attribute Type	Description	Req'd
Technology_Element_Name	String	Defines a technology element in which the FM is implemented.	Yes
Element_Name	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
Parent_Element	String	Connects the Failure Mode to its Parent in the FS hierarchy.	Yes
FMEDA_Name	String	Connects to the FMEDA project.	Yes
Element_Size_Permanent	Float [0,N]	Specifies the size of the Element for permanent faults for the associated Technology Element. This attribute is given precedence for an assumption-based FMEDA. Otherwise, the Element size is calculated based on the area extracted by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM. In the semiconductor world, this is the size of the logic implementing the intended functionality of the Element.	No
Element_Size_Transient	Float [0,N]	Specifies the size of the Element for transient faults for the associated Technology Element. This attribute is given precedence for an assumption-based FMEDA. Otherwise, the Element size is calculated based on the area extracted by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM.	No

		In the semiconductor world, this is the size of the logic implementing the intended functionality of the Element.	
Element_Size_Bits	Integer [0,N]	Specifies the size of the Element for transient and permanent faults for the associated Technology Element. This attribute is given precedence for an assumption-based FMEDA. Otherwise, the Element size is calculated based on the area extracted by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM. In the semiconductor world, this is the size of the memory logic included in the intended functionality of the Element.	No
Element_Mapping	List of Strings	Connects to the DUA representation and identifies the portion of the design implementing the intended functionality of the Element. This attribute is given precedence for a calculation-based FMEDA. Detailed semantics are to be defined in the LRM.	No
Element_Mapping_Exclude	List of Strings	Connects to the DUA representation and identifies the portion of the design to be excluded from the Element_Mapping. Can only be used in conjunction with the Element_Mapping attribute. This attribute is only used for a calculation-based FMEDA.	No
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

24 K. Define ISO26262 Failure Rate

Entity name FR_ISO26262

Key identifier Object_Name

Attribute Name	Attribute Type	Description	Req'd
FR_Type	Enumerate {Intrinsic_FR, SR_Failure_Rate, NSR_FR, Safe_FR, Non_Safe_FR, SPF_FR, Residual_FR, MPF_FR, MPF_Primary_FR, MPF_Secondary_FR, MPF_Detected, MPF_Perceived, MPF_Latent}	Failure Rates (FRs) calculated according to Figure 10, Part 10, Clause 8 of ISO26262 [2]	No
Scope	Enumerate {FMEDA, Element, Failure_Mode, Failure_Mode_Effect_Name}	Defines whether the FRs are calculated for the FMEDA, for an Element, or for a Failure Mode.	No
TE_Name	String	Specifies for which technology the FR is calculated.	No
Analysis_Type	Enumerate {Permanent, Transient}	Care to be taken about the effect of the FMEDA Analysis Type.	No
FMEDA_Name	String	Name of the FMEDA	No
Element_Name	String	This value is to be provided if the Scope = Element. This is used if the FR is combined at Element-level.	No
Parent_Element	String	Identifies he unique path for the Element	No
Failure_Mode	String	This value is to be provided if the Scope = Failure Mode	No

Failure_Mode_Effect	String	This value is to be provided if the Scope = Failure Mode Effect	No
Metric_Value	Float > 0	Value of the Failure Rate	No

26 L. Define ISO26262 Metric

Entity name Metric_ISO26262

Key identifier Object_Name

Attribute Name	Attribute Type	Description	Req'd
Metric_Name	Enumerate {SPFM, LFM, PMHF}	Metrics calculated according to ISO 26262.	No
Scope	Enumerate {FMEDA, Element, Failure_Mode, Failure_Mode_Effect_Name}	Defines whether the metrics are calculated for the FMEDA, for an Element, or for a Failure Mode.	No
TE_Name	String	Technology Element for which the metric is calculated	No
Analysis_Type	Enumerate {Permanent, Transient}	Care to be taken about the effect of FMEDA Analysis Type.	No
FMEDA_Name	String	Name of the FMEDA	No
Element_Name	String	This value is to be provided if the Scope = Element. This is used if the metric is combined at Element-level.	No
Parent_Element	String	Identifies the unique path for the Element.	No
Failure_Mode	String	This value is to be provided if the Scope = Failure Mode.	No
Failure_Mode_Effect	String	This value is to be provided if the Scope = Failure Mode Effect.	No

Metric_Value	Float > 0	Value of the metric	No
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

M. Define IEC61508 Failure Rate

Entity name FR_IEC61508

Key identifier Object_Name

Attribute Name	Attribute Type	Description	Req'd
FR_Type	Enumerate {Dangerous, Dangerous_Detected, Dangerous_Undetected }	Failure Rates (FRs) calculated according to IEC61508.	No
Scope	Enumerate {FMEDA, Element, Failure_Mode, Failure_Mode_Effect_Name }	Defines whether the FRs are calculated for the FMEDA, for an Element, or for a Failure Mode.	No
Analysis_Type	Enumerate {Permanent, Transient}	Care to be taken about the effect of the FMEDA Analysis Type.	No
FMEDA_Name	String	Name of the FMEDA	No
Element_Name	String	This value is to be provided if the Scope = Element. This is used if the FR is combined at the Element level.	No
Parent_Element	String	Identifies the unique path for the Element.	No
Failure_Mod	String	This value is to be provided if the Scope = Failure Mode.	No

e			
Failure_Mode_Effect	String	This value is to be provided if the Scope = Failure Mode Effect.	No
Metric_Value	Float > 0	Value of the Failure Rate	No
User_Defined_Attribute	List of tuples	List of previously created user-defined attributes and their values.	No

N. Define IEC61508 Metric

Entity name Metric_IEC61508

Key identifier Object_Name

Attribute Name	Attribute Type	Description	Req'd
Metric_Name	Enumerate {SFF, Probability_dangerous_failure_low_demand, Probability_dangerous_failure_high_demand}	Metrics calculated according to IEC61508.	No
Scope	Enumerate {FMEDA, Element, Failure_Mode, Failure_Mode_Effect_Name }	Defines whether the metrics are calculated for the FMEDA, for an Element, or for a Failure Mode.	No
FMEDA_Name	String	Name of the FMEDA	No
Element_Name	String	This value is to be provided if the Scope = Element. This is used if the FR is combined at the Element level.	No
Parent_Element	String	Identifies the unique path for the Element.	No
Failure_Mode	String	This value is to be provided if the Scope = Failure Mode.	No
Failure_Mode_Effect	String	This value is to be provided if the Scope = Failure Mode Effect.	No
Metric_Value	Float > 0	Value of the metric	No
User_Defined_	List of tuples	List of previously created user-defined attributes	No

Attribute		and their values.	

IX. Annex B – Language

A. Introduction

In this paper we defined a sample language for the only purpose of showing some concrete examples of usage of the Functional Safety Standard. The final LRM defined in the standard might differ from the sample used in this paper.

Following the principle of traceability, the sample language is derived directly from the conceptual data model with remarkably simple rules:

- Objects are created with “create” commands and updated with the “-update” option.
- Relationships are created with the "assign" commands.
- Weak objects are assigned a value with the "define" command.

In other words, the sample language is the implementation of the requirements defined in the conceptual data model.

A special rule stands for the Design mapping since it connects objects in the data model to objects in the design hierarchy, which are not part of the data model. The design mapping connection is described through the “-mapping” and “-exclude_mapping” options inside the design mapping relationship commands.

Table 12. Sample language derived from the data model.

FMEDA Process data	Entity type	Information type	Commands
FMEDA	FMEDA	Object	*create_fmeda*
FS Analysis Hierarchy	Element	Object	*create_element*
FM Hierarchy	Failure_Mode	Object	*create_fm*
Technology Element	Technology_Element	Object	*create_te*
Safety Mechanism Library	Safety_Mechanism	Object	*create_sm*
FM Effects	Failure_Mode_Effect	Object	*create_fme*
SM Mapping	SM-FM	Relationship	*assign_sm_fm*
FM Effect Mapping	FM-FME	Relationship	*assign_fm_fme*
TE Mapping	TE-FM	Relationship	*assign_te_fm*
TE Mapping	TE-Element	Relationship	*assign_te_element*
Design Mapping	Attribute of TE-FM	Relationship	*assign_te_fm -mapping {…}*
Design Mapping	Attribute of TE-Element	Relationship	*assign_te_element -mapping {…}*

ISO26262 Metrics	ISO26262_Metrics	Weak object	*define_metric_iso26262*
ISO26262 Failure Rate	ISO26262_FR	Weak object	*define_fr_iso26262*
IEC61508 Metrics	IEC61508_Metrics	Weak object	*define_metric_iec61508*
IEC61508 Failure Rate	IEC61508_FR	Weak object	*define_fr_iec61508*

The full list of commands defined according to the data model v0.1 is as follows:

- *create_fmeda*
- *create_element*
- *create_fm*
- *create_te*
- *create_sm*
- *create_fme*
- *add_attribute*
- *add_collection*
- *assign_sm_fm*
- *assign_fm_fme*
- *assign_te_fm*
- *assign_te_element*
- *define_fr_iso26262*
- *define_metric_iso26262*
- *define_fr_iec61508*
- *define_metric_iec61508*

As you can see, two commands are not directly derived from the FMEDA process: *add_attribute* and *add_collection*. These commands are auxiliary and serve the purpose of enabling reusability and extendibility of the data model and a language. If FMEDA project development can be limited solely to the objects and their attributes as defined by the data model, usage of *add_attribute* and *add_collection* is not necessary, although they can provide additional flexibility when needed.

B. Conventions

This document is using syntax highlight schema similar to IEEE 1801 (UPF) standard, chapter 5.2.

Key points can be summarized as follows:

- *italic* indicates user-defined variables
- [] square brackets indicate optional parameters
- {} curly braces indicate required values and can consist of one or more values
- <> angle brackets indicate a set of alternative parameters to choose from
- | separator bar indicates alternative choices within a group
- * asterisk indicates that a parameter can be repeated

Also, the "R" parameter of available arguments indicates a possibility to update a value of this argument using the -update switch.

For example, a *create_object* command (not a valid FS WG Language command) must be written as is, and it accepts a user defined value *object_name*; it also requires the *-dc* attribute to be specified. DC attributes can take multiple values as a list of lists, where the exact value is to be specified by the user.

create_object *object_name*
-dc { {<perm | tran | lat> *value %*}* }

Usage example:

```
create_object "SM_003"-dc {{perm 99} {tran 99}} -dc {lat 100}
```

The fact that the language visually appears to be relying on Tcl syntax doesn’t mean that the FS WG voted for Tcl or any other language to be the base interpreter language. In practice, this means that no assumptions regarding the usage of built-in Tcl (or any other language) constructs can be made, and virtually any language can be used to build a parser for a proposed language.

C. Safety Analysis Commands v0.1

Intentionally empty space.

101 create_fmeda

<table>
<tr><td>Purpose</td><td colspan="3">Create FMEDA project.</td></tr>
<tr><td>Syntax</td><td colspan="3">create_FMEDA FMEDA_name
[-type <assumption-based | calculation-based>]
[-asil [<a | b | c | d>]]
[-sil [<1 | 2 | 3 | 4>]]
[-analysis <permanent | transient | all>]
[-creator [{creator_name}]]
[-date [date]]
[-version version]
-data_model_version data_model_version
[-comment comment]
[-description description]
[-attribute { {name_of_the_user_defined_attribute value}* }]
-hierarchical [<yes | no>]
[-update]</td></tr>
<tr><td>Arguments</td><td>FMEDA_name</td><td>Name of the FMEDA project.</td><td></td></tr>
<tr><td></td><td>-data_model_version version</td><td>Version of the data model.</td><td></td></tr>
<tr><td></td><td>-type <assumption-based | calculation-based></td><td>Selects whether the FMEDA project is assumption-based or calculation-based. This attribute is informative only. If the type is calculation-based, the user can still specify the failure mode contribution through the "failure mode size" attribute.</td><td></td></tr>
<tr><td></td><td>-sil [<1 | 2 | 3 | 4>]</td><td>Defines the target safety level according to SIL classifications used by IEC61508 standards.</td><td>R</td></tr>
<tr><td></td><td>-asil [<a | b | c | d>]</td><td>Defines the target safety level according to ASIL classifications used by ISO26262 standards.</td><td>R</td></tr>
<tr><td></td><td>-analysis <permanent | transient | all></td><td>Defines the failure types to be considered and which metrics to be calculated within the safety analysis.

More than one value can be specified, e.g., Failure_Type = {Permanent} or Failure_Type = {Permanent, Transient}

The value “All” implies all Failure Types are activated. Defined as “All” instead of “Both” allows for plans for more than just Transient and Permanent.</td><td>R</td></tr>
<tr><td></td><td>-creator {creator_name}</td><td>Name of the company that generated the FMEDA.</td><td>R</td></tr>
<tr><td></td><td>-date date</td><td>Date when the FMEDA was generated.</td><td>R</td></tr>
<tr><td></td><td>-version fmeda_version</td><td>Version of the FMEDA project.</td><td>R</td></tr>
</table>

	-comment *comment*	Information that doesn't have a specific field in the FMEDA object.	R
	-description *description*	Description of the FMEDA project.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-hierarchical [<yes | no>]	Describes whether the FMEDA is fully flat or hierarchical, meant as aggregation of other FMEDAs. If no value is provided, then default **no** is used.	R
	-update	Indicates this command provides additional information for a previous command with the same *FMEDA_name*.	R
Return value	Return an empty string if successful or raise an ERROR if not.		

Usage example:

```
create_fmeda "Project_D" -type "assumption-based" -asil d -analysis all \
                         -creator "Hornet LLC" -date 27.01.2023 -version 0.1 \
                         -data_model_version 0.1 -hierarchical yes \
                         -comment "Project is an IP level project"

create_fmeda "Project_D" -type "calculation-based" -asil b -analysis perm \
                         -creator "Hornet LLC" -date 25.01.2023 -version 0.1 \
                         -data_model_version 0.1 -hierarchical yes \
                         -comment "Project is an IP level project" -update
```

create_element

Purpose	Create element.		
Syntax	create_element *element_name* -type <system | element | subelement | component | subcomponent | part | subpart> -fmeda *fmeda_name* [-description *description*] [-parent *parent*] [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*element_name*	Name (identifier) of the Element.	
	-type <system | element | subelement | component | subcomponent | part | subpart>	Specifies the type of the Element. Element_Type = Component or SubComponent can only be defined if the analysis is for IEC61508, inferred from the FMEDA entity, whether it has ASIL or SIL defined.	
	-fmeda *fmeda_name*	Connects the FS hierarchy to the FMEDA project.	R
	-description *description*	Description of the intended functionality of the Element.	R
	-parent *parent*	Connects the Element to its Parent in the FS hierarchy.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *element_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example:

```
create_element “A1” -type part -parent root -fmeda Project_A \
                    -description “Top-level element in a system”

create_element “A2” -type element -parent A1 -fmeda Project_A \
                    -description "Top-level element in a system"

create_element "A2" -type element -parent A1 -fmeda Project_A \
                    -description "2nd-level element in a system" \
                    -update
```

126 create_fm

<table>
<tr><td>Purpose</td><td colspan="3">Create failure mode.</td></tr>
<tr><td>Syntax</td><td colspan="3">create_fm *fm_name*
-parent *parent*
-fmeda *fmeda_name*
-type <mission | passive | active>
-safety_relevant [<yes | no>]
-dc_aggregation { { <perm | tran> <max | sum | residual | expert> }* }
-no_effect { { <perm | tran> *value %* }* }
-perceived { { <perm | tran> *value %* }* }
[-dc { {<perm | tran | lat> <calculated | measured> *value %* }* }]
[-safeness { {<perm | tran > <estimated | measured> *value %*}* }]
[-attribute { {*name_of_the_user_defined_attribute value*}* }]
[-description *description*]
[-update]</td></tr>
<tr><td>Arguments</td><td>*fm_name*</td><td>Name (identifier) of the Failure Mode.</td><td></td></tr>
<tr><td></td><td>-parent *parent*</td><td>Connects the Failure Mode to its Parent in the FS hierarchy.</td><td>R</td></tr>
<tr><td></td><td>-fmeda *fmeda_name*</td><td>Connects the FS hierarchy to the FMEDA project.</td><td>R</td></tr>
<tr><td></td><td>-type <mission | passive | active></td><td>Describes if and how the FM can violate a safety goal

Mission: participates to a safety function

Diagnostic is broken into:

- Passive: participates in a safety mechanism

- Active: participates in a safety mechanism that can violate a safety goal.</td><td>R</td></tr>
<tr><td></td><td>-safety_relevant [<yes | no>]</td><td>Specifies if the failure mode is safety related.

Safety_Relevant = no is equivalent to the "no part" according to IEC61508.</td><td>R</td></tr>
<tr><td></td><td>-dc_aggregation { { <perm | tran> <max | sum | residual | expert> }* }</td><td>Defines the heuristic/algorithm used to aggregate the DC of multiple SMs applied to the same FM. If DC_aggregation = expert, then the value is provided by the user using the attribute DC_expert.</td><td>R</td></tr>
<tr><td></td><td>-no_effect { { <perm | tran> *value %* }* }</td><td>No effect Permanent rate according to IEC 61508. This can only be used if the FMEDA has SIL target values (attribute of</td><td>R</td></tr>
</table>

		the FMEDA entity).	
	-perceived { { <perm | tran> *value %* }* }	Specifies that fraction of multi-point faults that are not detected but are perceived. This is only for ISO26262, i.e., if the FMEDA has the ASIL level defined.	R
	-dc { {<perm | tran | lat> <calculated | measured> *value %* [-attr_expr {*boolean_expr*}] }* }	Allows the user to specify the Permanent DC of the FM aggregated over the SMs associated with an FM. Only available if DC_Aggregation = expert.	R
	-safeness { {<perm | tran > <estimated | measured> *value %*}* }	Percentage/Fraction of safeness for permanent faults, i.e., faults that do not contribute to the violation of a safety goal.	R
	-attribute { {<_name_of_the_user_defined_attribute_ *value*}* }	Sets values of user-defined attributes.	R
	-description *description*	Description of the Failure Mode.	R
	-update	Indicates this command provides additional information for a previous command with the same *fm_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example:

```
set_scope {{"fmeda" "Project_A"} {"parent" "A1"}
    create_fm "FM001" -type mission -safety_relevant yes -dc_aggregation max \
        -no_effect { {perm 90} {tran 90} }
        -perceived { {perm 50} {tran 50} }
        -dc { {perm measured 99.5} {tran measured 99.5} {lat measured 99.5} }
        -dc { {perm calculated 99.5} {tran calculated 99.5} {lat calculated 99.5}
}
        -safeness { {perm measured 99.5} {tran measured 99.5} }
        -safeness { {perm estimated 99.5} {tran estimated 99.5} }
        -description "Some random overcomplicated FM"

    create_fm "FM001" -type mission -safety_relevant yes -dc_aggregation max \
        -no_effect { {perm 90} {tran 90} }
        -perceived { {perm 50} {tran 50} }
        -dc { {perm measured 99.5 -attr_expr {config == "b"}} \
                {tran measured 99.5 -attr_expr {config == "b"}} \
                {lat measured 99.5 -attr_expr {config == "b"}} }
        -dc { {perm calculated 99.5} {tran calculated 99.5} {lat calculated 99.5}
}
        -safeness { {perm measured 99.5} {tran measured 99.5} }
        -safeness { {perm estimated 99.5} {tran estimated 99.5} }
        -description "Some random FM" -update
```

151 create_te

Purpose	Create technology element.		
Syntax	create_te *te_name* -type [<digital | ram | rom | flash | analog | custom>] [-source [<IEC_62380 | SN_25900 | expert>] -fr { {<perm | tran > *value*}* } [-fr_derating *value %*] -unit_design_element_size *value* [-description *description*] [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*te_name*	Name (identifier) of the Technology_Element.	
	-type [<digital | ram | rom | flash | analog | custom>]	Name of the technology type for a given technology element.	R
	-source [<IEC_62380 | SN_25900 | expert>]	Description of the source of BFR data (e.g., IEC TR 62380, testing, field returns …).	R
	-fr { {<perm | tran > *value*}* }	Base Failure Rate (BFR) for permanent or transient faults.	R
	-fr_derating *value*	Derating of the BFR for transient faults in digital technology elements. Used to account for the contribution of combinatorial logic to the raw fit transient (as a percentage of the raw fit transient from sequential/memory elements).	R
	-unit_design_element_size *value*	Area of the unit design element of the technology element. Required if Technology_Element = Digital/Analog. Not utilized if Technology_Element = RAM/ROM/Flash. It is used to calculate the number of unit design elements in an FM and hence calculates the raw FIT for the FM. The following formula applies: `*#FM_unit_design_elements = FM_size_permanent/unit_design_element_size*`. It should be set to 1 if the *FM_size_permanent/FM_size_transient* is already expressed in number of unit design elements.	R
	-description *description*	Description of the technology element.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *te_name*.	R

Return value	Returns an empty string if successful or raises an ERROR if not.

Usage example:

```
Create_te “Digital_Area” -type digital -source IEC_62380 -fr {{perm 0.03033} {tran
0}     }
create_te “Analog_Area”  -type analog  -source IEC_62380 -fr {{perm 0.03033} {tran
0.01}  }
create_te “ROM”          -type ram     -source IEC_62380 -fr {{perm 0.03033} {tran
1e-7}  }
```

161 create_sm

<table>
<tr><td>Purpose</td><td colspan="3">Create safety mechanism.</td></tr>
<tr><td>Syntax</td><td colspan="3">create_sm sm_name
[-fmeda fmeda_name]
[-class [<HW | SW | AoU | AoU-SW | AoU-HW | user-defined>]]
[-class_description class_description]
-configurable [<yes | no>]
-dc { {<perm | tran | lat> value %}* }
[-description description]
[-attribute { {name_of_the_user_defined_attribute value}* }]
[-update]</td></tr>
<tr><td rowspan="8">Arguments</td><td>sm_name</td><td>Name (identifier) of the Safety Mechanism.</td><td></td></tr>
<tr><td>-fmeda fmeda_name</td><td>Connects the FS hierarchy to the FMEDA project.</td><td>R</td></tr>
<tr><td>-class [<HW | SW | AoU | AoU-SW | AoU-HW | user-defined>]</td><td>Method by which the safety mechanism is to be realized.

Notes:
1) AoU is to capture when the SM is not part of the product (potentially raise a flag during FMEDA integration).
2) HW allows for further specification for downstream tools.</td><td>R</td></tr>
<tr><td>-class_description class_description</td><td>Description of the class.

Note: Especially meant in the case in which the class is user-defined, but is available for all classes.</td><td>R</td></tr>
<tr><td>-configurable [<yes | no>]</td><td>Captures whether the SM can be turned on or off by the user/integrator. If configurable=yes, then the “SM-FM active” attribute can be used.</td><td>R</td></tr>
<tr><td>-dc { {<perm | tran | lat> value %}* }</td><td>Diagnostic coverage of the SM in isolation for permanent faults.

Notes: To apply a diagnostic coverage specific to an SM-FM pair, use the DC_perm attribute in the SM-FM category. When SM:DC_perm and SM-FM:DC_perm are specified, the SM-FM:DC_perm attribute takes precedence.</td><td>R</td></tr>
<tr><td>-description description</td><td>Description of the intended functionality of the SM.</td><td>R</td></tr>
<tr><td>-attribute { {name_of_the_user_defined_attribute value}* }</td><td>Sets values of user-defined attributes.</td><td>R</td></tr>
</table>

	-update	Indicates this command provides additional information for a previous command with the same *sm_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example:

```
create_sm "SM_001" -class "AoU-SW" -collection "SM_default_99"

create_sm "SM_001.5" -class "AoU" -collection "SM_default_99" -class_description
"What exactly do we assume?.."

create_sm "SM_002" -fmeda "CPU_FMEDA" -class "HW" -configurable "no" -dc {{perm
99} {tran 99} {lat 100}}

create_sm "SM_003" -class "HW" -configurable "no" -dc {{perm 99} {tran 99}} -dc
{lat 100}
create_sm "SM_003" -class "AoU-HW" -configurable "no" -dc {{perm 99} {tran 99}} -
dc {lat 100} -update
```

create_fme

Purpose	Create failure mode effect.		
Syntax	create_fme *fme_name* -fmeda *fmeda_name* [-description *description*] [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*fme_name*	Name (identifier) of the Failure Mode effect.	
	-fmeda *fmeda_name*	Connects the FME to the FMEDA project.	R
	-description *description*	Description of the FME.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *fme_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example:

```
create_fme “FME001” -fmeda IP_A -description “Loss of data”
create_fme “FME002” -fmeda IP_A -description “Incorrect data”
```

add_attribute

Purpose	Create new attribute.		
Syntax	add_attribute *attribute_name* -default *value* -fmeda *fmeda_name* [-object [<global | *entity_object* >]] [-type [{ <string | int [*min max*] | float [*min max*] | enum {*list_of_enum_values*} > }]] [-description *description*] [-update]		
Arguments	*te_name*	Name (identifier) of the attribute.	
	-default *value*	Default value of the attribute.	R
	-fmeda *fmeda_name*	Connects to the FMEDA project.	R
	-object [<global | *entity_object* >]	Defines an object of an ERD on which to enable the use of user-defined attributes.	R
	-type [{ <string | int [*min max*] | float [*min max*] | enum {*list_of_enum_values*} > }]	Type hinting for the tools' backend. Enables the tool to check a type of the attribute (similar to system-defined attribute).	R
	-description *description*	Description of the attribute.	R
	-update	Indicates this command provides additional information for a previous command with the same *te_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

This command uses the individual *set -attribute* command to work with built-in attributes of safety objects. It works with user-defined attributes, inspired by the asciidoctor text markup toolchain that is using custom attributes of objects. All attributes are defined within a single *create_** command. In addition, efficient usage of UDA (user-define attributes) is heavily linked to the proposed new extension of the *-attr_expr* command that is inspired by UPF's -*logic_expr*. Together these features enable rich reconfigurability and extendibility of the proposed language and can mimic functionalities that the safety community is used to (e.g., creating a new column in a spreadsheet).

In addition to reconfigurability, this command also enables various users to store extra information inside the data model itself.

Similar to the RISC-V community, here it is also expected that users will contribute to this WG and share their feedback regarding their most used custom user-defined attributes so that the WG can potentially introduce those attributes as built-ins in a later release of the language. If this expectation is not satisfied, there can be a vendor-lock for certain features, similar to custom attributes and pragmas in SystemVerilog.

Usage example:

```
add_attribute "strobing_point" -object "fm" -default ""
add_attribute "config" -object "create_sm" -default ""

create_fm "FM_001" -parent "MULT16" \
                   -attribute { "strobing_point" "top.SoC.IP1.IP2.output_x" }

create_sm "SM_001" -class "AoU-SW" -configurable "no" \
                   -dc {{perm 90} {tran 90} {lat 100}} \
                   -attribute { "config" {"ASIL_D_CONFIG" "ASIL_B_CONFIG"
"QM_CONFIG"} }
```

- attribute - user-defined attribute

Usage example with a multiple definition attempt:

```
add_attribute "Diagnostic or Avoidance" -object "fm" -default "Avoidance"
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance"
```

- Error upon execution: To enable a user-defined attribute on multiple ERD entities, use the *"global" -object*.

Usage example with type hinting:

```
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance" -type
{enum {"Avoidance" "Diagnostic"}}

add_collection "Baseline SM values" -object "sm" \
    -list { {"Diagnostic or Avoidance" "Undefined"} \
           }
```

- Error upon execution: User-defined attribute “Diagnostic or Avoidance” of type “enum” does not support the "Undefined" input value.

Usage example with type hinting:

```
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance" -type
{enum {"Avoidance" "Diagnostic"}}
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance" -type
"string" -update
```

- Error upon execution: Cannot update a user-defined attribute’s datatype.

Usage example: Extend the data model using UDA to enable new tool-level features.

add_collection

Purpose	Create a new collection of attributes.	
Syntax	add_collection *collection_name* -object *entity_object* -list { {*name_of_the_attribute value*}* } -fmeda *fmeda_name* [-description *description*]	
Arguments	*collection_name*	Name (identifier) of the attribute.
	-object *entity_object*	The type of the attribute limits its applicability to various objects of ERD.
	-list { {*name_of_the_attribute value*}* }	List of lists with defined names and values of the attributes of the selected ERD object.
	-fmeda *fmeda_name*	Connects to the FMEDA project.
	-description *description*	Description of the attribute.
Return value	Returns an empty string if successful, or raises an ERROR if not.	

Notes:

- Collection works as an intermediate storage of attribute-value pairs before they get assigned to an ERD object, weak object, or a relationship.
- Collection cannot be updated.
- Collection cannot be redefined.
    - Repetitive declarations are to be discarded.
- Collection cannot be empty.
- Collection can use previously defined user-defined attributes.
- Collection must belong to particular ERD object.
- Collection cannot use attributes that do not belong to the selected ERD entity.
- Collection, when connected to a safety object, cannot overwrite attributes' values already stored in an ERD entity.
- Values of attributes defined in a collection are copied over to an ERD entity upon connection of said ERD object to the collection. Connection to be done by additional key *-collection*.

Usage example with Safety Mechanisms definition:

```
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance"
add_attribute "Error Response"          -object "sm" -default "HW Error Flag"
```

```
add_attribute "ISO26262 DC"             -object "sm" -default "High"

add_collection "Baseline SM values" -object "sm" \
    -list { {"Diagnostic or Avoidance" "Diagnostic"} \
            {"Error Response" "Diagnostic"} \
            {"ISO26262 DC" "High"} \
            {"dc" {perm 95}} \
            {"dc" {tran 90}} \
            {"dc" {lat 0}} \
            {"configurable" "no"} \
            {"class" "HW"} \
            {"fmeda" "TOP"} \
          }
    -fmeda TOP


create_sm "SM_001" -collection "Baseline SM values" -description "My first SM, with default values assigned"
create_sm "SM_002" -collection "Baseline SM values" -description "My second SM, with default values assigned"
create_sm "SM_003" -collection "Baseline SM values" -description "My third SM, with default values assigned"
create_sm "SM_004" -collection "Baseline SM values" -description "My 4th SM, with default values assigned"

## Equivalent single command:
create_sm "SM_005" -class "HW" -configurable "no" -dc {{perm 95} {tran 90} {lat 0}} \
    -attribute { {"Diagnostic or Avoidance" "Diagnostic"} \
                 {"Error Response" "Diagnostic"} \
                 {"ISO26262 DC" "High"} \
               } \
    -fmeda "TOP"
    -description "My 5th SM, with values assigned explicitly"
```

Usage example with redefinition attempt:

```
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance"
add_attribute "Error Response"          -object "sm" -default "HW Error Flag"

add_collection "Baseline SM values" -object "sm" \
    -list { {"Diagnostic or Avoidance" "Diagnostic"} \
            }


add_collection "Baseline SM values" -object "sm" \
    -list { {"Diagnostic or Avoidance" "Avoidance"} \
            }
```

- Error upon execution: Cannot redefine existing "Baseline SM values" collection.

Usage example with type mismatch:

```
add_attribute "Diagnostic or Avoidance" -object "fm" -default "Avoidance"

add_collection "Baseline SM values" -object "sm" \
```

```
 -list { {"Diagnostic or Avoidance" "Diagnostic"} \
       }
```

- Error upon execution: Illegal access to the user-defined "Diagnostic or Avoidance" attribute. Type mismatch: The attribute "Diagnostic or Avoidance" belongs to the "fm" ERD entity, whereas the collection belongs to the "sm" ERD entity.

314 assign_sm_fm

<table>
<tr><td>Purpose</td><td colspan="3">Assign safety mechanism to failure mode.</td></tr>
<tr><td>Syntax</td><td colspan="3">assign_sm_fm smfm_name
-sm_name safety_mechanism
-fm_name failure_mode
-parent parent
-fmeda fmeda_name
[-dc { {<perm | tran | lat> <estimated | measured> value %}* }]
-active <yes | no>
[-attribute { {name_of_the_user_defined_attribute value}* }]
[-update]</td></tr>
<tr><td rowspan="9">Arguments</td><td>smfm_name</td><td>Name (identifier) of the assignment.</td><td></td></tr>
<tr><td>-sm_name safety_mechanism</td><td>Name (identifier) of the SM applied to the FM.</td><td>R</td></tr>
<tr><td>-fm_name failure_mode</td><td>Non-unique name (identifier) of the FM covered by the SM.</td><td>R</td></tr>
<tr><td>-parent parent</td><td>Defines a parent scope for a previously defined Failure Mode to make an FM definition unambiguous.</td><td>R</td></tr>
<tr><td>-fmeda fmeda_name</td><td>Connects to the FMEDA project.</td><td>R</td></tr>
<tr><td>-dc { {<perm | tran | lat> <estimated | measured> value %}* }</td><td>Diagnostic coverage of the SM applied to the FM for permanent faults. If no value is specified, the DC_Perm value of the SM entity will be used.

Notes: This value is specific to the SM-FM pair and takes precedence over the DC_perm of the SM category. If this is not specified, then the value is taken from the DC_perm attribute of the SM category.</td><td>R</td></tr>
<tr><td>-active [<yes | no>]</td><td>Specifies whether the SM is enabled for this FM.

Only accessible if the SM_Configurable attribute = yes.</td><td>R</td></tr>
<tr><td>-attribute { {name_of_the_user_defined_attribute value}* }</td><td>Sets values of user-defined attributes.</td><td>R</td></tr>
<tr><td>-update</td><td>Indicates this command provides additional information for a previous command with the same smfm_name.</td><td>R</td></tr>
<tr><td>Return value</td><td colspan="2">Returns an empty string if successful, or raises an ERROR if not.</td><td></td></tr>
</table>

Usage example:

```
add_collection "SM_default_99" -object "sm" -fmeda "CPU_FMEDA" \
    -list { {"configurable" "no"} \
            {"dc" "{{perm 99} {tran 99} {lat 100}}"} \
            {"fmeda" "CPU_FMEDA"} }

create_sm "SM_001" -class "AoU-SW" -collection "SM_default_99"

assign_sm_fm "SM_001_to_ALU_X.MULT32.FM001" -sm_name "SM_001" -fm_name "FM_001" -
parent "ALU_X.MULT32" \
    -fmeda  "CPU_FMEDA" -attribute {"use_case_generic" "no"} \
    -dc {{perm estimated 90} {tran estimated 90} {lat estimated 100}}
```

329 assign_fm_fme

Purpose	Assign failure mode to failure mode effect.		
Syntax	assign_fm_fme *fmfme_name* -fm_name *failure_mode* -parent *parent* -fme_name {fme_*name* } -fmeda *fmeda_name* -fme_weight { *fme_weight* } [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*fmfme_name*	Name (identifier) of the assignment.	
	-fm_name *failure_mode*	Name (identifier) of the FM contributing to the FME.	R
	-parent *parent*	Defines a parent scope for a previously defined Failure Mode to make an FM definition unambiguous.	R
	-fme_name { *fme_name* }	List of names (identifiers) of the FMEs caused by the FM. Connects the FM to the FME that represents the consequence seen at the top level (of the DUA scope).	R
	-fmeda *fmeda_name*	Connects to the FMEDA project.	R
	-fme_weight { *fme_weight* }	Weights of the contributions of the FM to the list of FMEs defined in FME_list.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *fmfme_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

330

331 assign_te_fm

<table>
<tr><td>Purpose</td><td colspan="3">Assign technology element to failure mode.</td></tr>
<tr><td>Syntax</td><td colspan="3">assign_te_fm tefm_name
-te_name { te_name }
-fm_name { fm_name }
-parent parent
-fmeda fmeda_name
-fm_size { {<percentage | absolute | uniform-distribution> <perm | tran | bits> value }* }
[-fm_mapping { {<sv | vhdl | spice | user-defined> path }* }]
[-fm_mapping_exclude { {<sv | vhdl | spice | user-defined> path }* }]
[-attribute { {name_of_the_user_defined_attribute value}* }]
[-update]</td></tr>
<tr><td>Arguments</td><td>tefm_name</td><td>Name (identifier) of the assignment.</td><td></td></tr>
<tr><td></td><td>-te_name { te_name }</td><td>Defines a technology element in which the FM is implemented.</td><td>R</td></tr>
<tr><td></td><td>-fm_name { fm_name }</td><td>Defines a name of the target failure mode.</td><td>R</td></tr>
<tr><td></td><td>-parent parent</td><td>Connects the Failure Mode to its Parent in the FS hierarchy.</td><td>R</td></tr>
<tr><td></td><td>-fmeda fmeda_name</td><td>Connects to the FMEDA project.</td><td>R</td></tr>
<tr><td></td><td>-fm_size { {<percentage | absolute | uniform-distribution> <perm | tran | bits> value }* }</td><td>The first value of an array defines whether the FM_Size will be:

• Percentage: A percentage of the parent Element_Size
• Absolute: An absolute value
• Uniform_Distribution: A uniform distribution of the parent Element_Size

The second value of an array defines the type of faults that can occur:

• Permanent
• Transient
• Bit

The third value defines the size of a FM where a given type of fault can occur. This is used to calculate a FMD for the associated TE. In the semiconductor world, these fault types are associated with combinatorial and sequential logic gates, sequential logic gates, and storage elements respectively.

This attribute is given precedence for an assumption-based FMEDA. Otherwise, the FMD is calculated based on the area of the FM defined by the mapping to the design</td><td>R</td></tr>
</table>

<table>
<tr><td></td><td></td><td>hierarchy. Detailed semantics are to be defined in the LRM.</td><td></td></tr>
<tr><td></td><td>-fm_mapping { {<sv | vhdl | spice | user-defined> path }* }</td><td>Connects to the DUA representation and identifies the portion of the design responsible for the Failure Mode. This attribute is given precedence for a calculation-based FMEDA. Detailed semantics are to be defined in the LRM.</td><td>R</td></tr>
<tr><td></td><td>-fm_mapping_exclude { {<sv | vhdl | spice | user-defined> path }* }</td><td>Connects to the DUA representation and identifies the portion of the design to be excluded from the FM_Mapping. Can only be used in conjunction with the FM_Mapping attribute. This attribute is only used for a calculation-based FMEDA.</td><td>R</td></tr>
<tr><td></td><td>-attribute { {name_of_the_user_defined_attribute value}* }</td><td>Sets values of user-defined attributes.</td><td>R</td></tr>
<tr><td></td><td>-update</td><td>Indicates this command provides additional information for a previous command with the same tefm_name.</td><td>R</td></tr>
<tr><td>Return value</td><td colspan="3">Returns an empty string if successful, or raises an ERROR if not.</td></tr>
</table>

Usage example:

```
create_fm "FM_001" -parent "ELEMENT_A.ELEMENT_B" -safety_relevant "no"

assign_fm_fme "TE_002" -source "expert" -fr {{perm 0.00000000073} {tran
0.00000000019}}

assign_te_fm "AT_006" -te_name "TE_002" --fm_name "FM_001" -parent
"ELEMENT_A.ELEMENT_B" \
    -fm_mapping {top.c.u.g.*, top.c.x.p.v.*, top.f.n.*} \
    -fm_size {{absolute perm 26.45} {absolute tran 1428.73}}
```

344 assign_te_element

<table>
<tr><td>Purpose</td><td colspan="3">Assign technology element to element.</td></tr>
<tr><td>Syntax</td><td colspan="3">assign_te_element teelement_name
-te_name { te_name }
-element_name { element_name }
-parent parent
-fmeda fmeda_name
[-element_size { {<perm | tran | bits> value }* }]
[-element_mapping { path }]
[-element_mapping_exclude { path }]
[-attribute { {name_of_the_user_defined_attribute value}* }]
[-update]</td></tr>
<tr><td rowspan="8">Arguments</td><td>teelement_name</td><td>Name (identifier) of the assignment.</td><td></td></tr>
<tr><td>-te_name { te_name }</td><td>Defines a technology element in which the FM is implemented.</td><td>R</td></tr>
<tr><td>-element_name { element_name }</td><td>Defines an element to be connected to a Technology element.</td><td>R</td></tr>
<tr><td>-parent parent</td><td>Connects the Element to its Parent in the FS hierarchy.</td><td>R</td></tr>
<tr><td>-fmeda fmeda_name</td><td>Connects to the FMEDA project.</td><td>R</td></tr>
<tr><td>-element_size { {<perm | tran | bits> value }* }</td><td>The first value of an array defines the type of faults that can occur:
Permanent
Transient
Bit
The second value defines the size of an element where a given type of fault can occur for the corresponding TE.

This attribute is given precedence for an assumption-based FMEDA. Otherwise, the Element size is calculated based on the area extracted by the mapping to the design hierarchy. Detailed semantics are to be defined in the LRM.</td><td>R</td></tr>
<tr><td>-element_mapping { path }</td><td>Connects to the DUA representation and identifies the portion of the design implementing the intended functionality of the Element. This attribute is given precedence for a calculation-based FMEDA. Detailed semantics are to be defined in the LRM.</td><td>R</td></tr>
<tr><td>-element_mapping_exclude { path }</td><td>Connects to the DUA representation and identifies the portion of the design to be</td><td>R</td></tr>
</table>

		excluded from the Element_Mapping. Can only be used in conjunction with the Element_Mapping attribute. This attribute is only used for a calculation-based FMEDA.	
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *teelement_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example:

```
create_te "Analog_5n"  -type "analog"  -fr {perm  3e-9}

create_element "PARTN" -fmeda "TOP" -type element
create_element "S_PART_X" -fmeda "TOP" -type element -parent "PARTN"

assign_te_element -fmeda "TOP" -te_name "Analog_5n" -element_name "S_PART_Y" -
parent "PARTN.S_PART_X"  \
    -element_mapping {top.a.b.c.*, top.a.b.p.q.*, top.a.s.t.p.*}
    -element_size {{perm 582.18} {tran 438.21} {bits 512}}
```

357 **define_fr_iso26262**

Purpose	Define a value of a Failure Rate associated with particular scope according to the FR type.		
Syntax	define_fr_iso26262 *fr_name* -fr_type { {<intrinsic_fr | sr_failure_fr | nsr_fr | safe_fr | non_safe_fr | spf_fr | residual_fr | mpf_fr | mpf_primary_fr | mpf_seconday_fr | mpf_detected | mpf_perceived | mpf_latent > *fr_value* }* } -scope { <fmeda | element | fm | fme> *value* [*parent*] } [-te_name { *te_name* }] -analysis_type <perm | tran> -fmeda *fmeda_name* [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*fr_name*	Name (identifier) of the failure rate.	
	-fr_type { {<intrinsic_fr | sr_failure_fr | nsr_fr | safe_fr | non_safe_fr | spf_fr | residual_fr | mpf_fr | mpf_primary_fr | mpf_seconday_fr | mpf_detected | mpf_perceived | mpf_latent > *fr_value* }* }	Failure Rates (FR) calculated according to Figure 10, Part 10, Clause 8 of ISO26262 [2].	R
	-scope { <fmeda | element | fm | fme> *value* [*parent*] }	Defines whether the FRs are calculated for the FMEDA, for an Element, for a Failure Mode, or for a Failure Mode Effect.	R
	-te_name { *te_name* }	Specifies for which technology the FR is calculated.	R
	-analysis_type <perm | tran>	Specifies the analysis type the calculated FR belongs to.	R
	-fmeda *fmeda_name*	Connects to the FMEDA project.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *fr_name*.	R

358

define_metric_iso26262

<table>
<tr><td>Purpose</td><td colspan="3">Define a value of a Metric associated with a particular scope according to the Metric's type.</td></tr>
<tr><td>Syntax</td><td colspan="3">define_metric_iso26262 metric_name
-metric_type { {<spfm | lfm | pmhf> metric_value }* }
-scope { <fmeda | element | fm | fme> value [parent] }
-te_name { te_name }
-analysis_type <perm | tran>
-fmeda fmeda_name
[-attribute { {name_of_the_user_defined_attribute value}* }]
[-update]</td></tr>
<tr><td>Arguments</td><td>metric_name</td><td>Name (identifier) of the metric definition.</td><td></td></tr>
<tr><td></td><td>-metric_type { {<spfm | lfm | pmhf> metric_value }* }</td><td>Metrics calculated according to ISO 26262 [2].</td><td>R</td></tr>
<tr><td></td><td>-scope { <fmeda | element | fm | fme> value [parent] }</td><td>Defines whether the metrics are calculated for the FMEDA, for an Element, for a Failure Mode, or for a Failure Mode Effect.</td><td>R</td></tr>
<tr><td></td><td>-te_name { te_name }</td><td>Specifies for which technology the FR is calculated.</td><td>R</td></tr>
<tr><td></td><td>-analysis_type <perm | tran></td><td>Care to be taken about the effect of FMEDA Analysis Type. (Pending the decision on whether we will have a single language for input+output or two separate ones.)</td><td>R</td></tr>
<tr><td></td><td>-fmeda fmeda_name</td><td>Connects to the FMEDA project.</td><td>R</td></tr>
<tr><td></td><td>-attribute { {name_of_the_user_defined_attribute value}* }</td><td>Sets values of user-defined attributes.</td><td>R</td></tr>
<tr><td></td><td>-update</td><td>Indicates this command provides additional information for a previous command with the same metric_name.</td><td>R</td></tr>
<tr><td>Return value</td><td colspan="3">Returns an empty string if successful, or raises an ERROR if not.</td></tr>
</table>

Usage example:

```
define_metric_iso26262 SPFM_Measured_P_global -metric_type {spfm 91.96} -scope
{fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -analysis_type perm
define_metric_iso26262 SPFM_Measured_T_global -metric_type {spfm 97.95} -scope
{fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -analysis_type tran
define_metric_iso26262 LFM_Measured_T_global  -metric_type {lfm 92.74}  -scope
{fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -analysis_type perm
define_metric_iso26262 PMHF_Measured_P_global -metric_type {pmhf 4.970} -scope
{fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -analysis_type perm
```

```
define_metric_iso26262 PMHF_Measured_T_global -metric_type {pmhf 1.786E-6} -scope
{fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -analysis_type tran

define_metric_iso26262 IP_A_Global_Perm -metric_type {{spfm 91.96} {lfm 92.74}
{pmhf 4.970}}-scope {fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -
analysis_type perm
define_metric_iso26262 IP_A_Global_Tran -metric_type {{spfm 97.95} {pmhf 1.786E-
6}} -scope {fmeda IP_A} -te_name {"Digital_Area" "RAM"} -fmeda IP_A -analysis_type
tran
```

380 ## define_fr_iec61508

Purpose	Define the value of a Failure Rate associated with a particular scope according to the FR type.		
Syntax	define_fr_iec61508 *fr_name* -fr_type { {<dangerous | dangerous_detected | dangerous_undetected > *fr_value* }* } -scope { <fmeda | element | fm | fme> *value* [*parent*] } [-te_name { *te_name* }] -analysis_type <perm | tran> -fmeda *fmeda_name* [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*fr_name*	Name (identifier) of the failure rate.	
	-fr_type { {<intrinsic_fr | sr_failure_fr | nsr_fr | safe_fr | non_safe_fr | spf_fr | residual_fr | mpf_fr | mpf_primary_fr | mpf_seconday_fr | mpf_detected | mpf_perceived | mpf_latent > *fr_value* }* }	Failure Rates (FR) calculated according to IEC 61508 [3].	R
	-scope { <fmeda | element | fm | fme> *value* [*parent*] }	Defines whether the FRs are calculated for the FMEDA, for an Element, for a Failure Mode, or for a Failure Mode Effect.	R
	-te_name { *te_name* }	Specifies for which technology the FR is calculated.	R
	-analysis_type <perm | tran>	Care to be taken about the effect of the FMEDA Analysis Type. (Pending the decision on whether we will have a single language for input+output or two separate ones.)	R
	-fmeda *fmeda_name*	Connects to the FMEDA project.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *fr_name*.	R

381

define_metric_iec61508

<table>
<tr><td>Purpose</td><td colspan="3">Define the value of a Metric associated with a particular scope according to the Metric’s type.</td></tr>
<tr><td>Syntax</td><td colspan="3">define_metric_iec61508 metric_name
-metric_type { {<SFF | Probability_dangerous_failure_low_demand | Probability_dangerous_failure_high_demand> metric_value }* }
-scope { <fmeda | element | fm | fme> value [parent] }
-te_name { te_name }
-analysis_type <perm | tran>
-fmeda fmeda_name
[-attribute { {name_of_the_user_defined_attribute value}* }]
[-update]</td></tr>
<tr><td rowspan="8">Arguments</td><td>metric_name</td><td>Name (identifier) of the metric definition.</td><td></td></tr>
<tr><td>-metric_type { {<SFF | Probability_dangerous_failure_low_demand | Probability_dangerous_failure_high_demand> metric_value }* }</td><td>Metrics calculated according to IEC 61508 [3].</td><td>R</td></tr>
<tr><td>-scope { <fmeda | element | fm | fme> value [parent] }</td><td>Defines whether the metrics are calculated for the FMEDA, for an Element, for a Failure Mode, or for a Failure Mode Effect.</td><td>R</td></tr>
<tr><td>-te_name { te_name }</td><td>Specifies for which technology the FR is calculated.</td><td>R</td></tr>
<tr><td>-analysis_type <perm | tran></td><td>Care to be taken about the effect of the FMEDA Analysis Type. (Pending the decision on whether we will have a single language for input+output or two separate ones.)</td><td>R</td></tr>
<tr><td>-fmeda fmeda_name</td><td>Connects to the FMEDA project.</td><td>R</td></tr>
<tr><td>-attribute { {name_of_the_user_defined_attribute value}* }</td><td>Sets values of user-defined attributes.</td><td>R</td></tr>
<tr><td>-update</td><td>Indicates this command provides additional information for a previous command with the same metric_name.</td><td>R</td></tr>
<tr><td>Return value</td><td colspan="3">Returns an empty string if successful, or raises an ERROR if not.</td></tr>
</table>

X. Annex C – Add-on to v0.1

This chapter describes commands that were considered by the working group, but no decision was agreed on whether accept or decline them. This chapter is for informative purposes only.

The full list of commands defined according to this extension is as follows:

- *load_slf*
- *save_slf*
- *set_scope*
- *add_parameter*
- *attr_expr*
- *assign_fmeda_fmeda*
- *assign_fmeda_element*

load_slf

SLF = safety language format. This naming was created to enable users to write scripts in SLF and show examples containing file extensions.

This naming is not approved by the WG.

Purpose	Load a project described with the language defined by Accellera's FS WG.	
Syntax	load_slf *filename* [-prefix *name_of_the_prefix*] [-parameters { { *name_of_the_parameter value*}* }]	
Arguments	*filename*	Name of the file to load into the Tcl console.
	-prefix *name_of_the_prefix*	A text value to prepend to all objects within a loaded file.
	-parameters { { *name_of_the_parameter value*}* }	Overwrites values of parameters defined in the loaded file.
Return value	Returns an empty string if successful, or raises an ERROR if not.	

Usage example:

```
######################################################
### SoC Project in ASIL D configuration ##############
######################################################
load_slf "Project_A.slf" -parameters { "ATTR_ASIL_LEVEL" d }

######################################################
### SoC Project in ASIL B configuration ##############
######################################################
add_parameter "ATTR_ASIL_LEVEL2" -default d
load_slf "Project_B.slf"
```

- Parameter *ATTR_ASIL_LEVEL* has a scope of *load_slf* command only.
- Parameter *ATTR_ASIL_LEVEL2* has a global scope, including the *load_slf* command.

save_slf

SLF = safety language format. This naming was created to enable users to write scripts in SLF and show example containing file extensions.

This naming is not approved by the WG.

Purpose	Save active project in a target tool as a project in SLF format.	
Syntax	save_slf *filename* [-fmeda *fmeda_name*]	
Arguments	*filename*	Name of the file to save to.
	-fmeda *fmeda_name*	Name of the project to save. If omitted, all available projects are to be saved.
Return value	Returns an empty string if successful, or raises an ERROR if not.	

set_scope

Purpose	Set the scope of execution for subsequent commands.
Syntax	set_scope [{ { <fmeda | parent | parent_prefix> *value* }* }]
Arguments	{ { <fmeda | parent | parent_prefix> *value* }* } — Sets the value of the **-fmeda** key for all subsequent calls. Sets the value of a **-parent** key for all subsequent calls. Sets the value of a prefix for a **-parent** key for all subsequent calls. An empty value resets all scoping settings.
Return value	Returns an empty string if successful, or raises an ERROR if not.

Usage examples:

```
create_fmeda "Project_D"
set_scope {"fmeda" "Project_D"}
  create_element  "D1" -type "part" -parent "root"
    create_fm     "FM001"            -parent   "D1" -dc {"tran" "measured" 91.5}
    create_fm     "FM002"            -parent   "D1" -dc {"tran" "measured" 91.4}
  create_element sD1 -type "subpart" -parent  "D1"
  set_scope {"parent" "sD1"}
    create_fm     "FM003"  -dc {"tran" "measured" 71.5}
    create_fm     "FM004"  -dc {"tran" "measured" 71.4}
  set_scope {{"parent" ""} {"parent_prefix" "D1.sD1"}
    create_element  C1 -type    "part" -parent root
      create_fm     "FM001"              -parent  "C1" -dc {"perm" "measured" 99.5}
      create_fm     "FM002"              -parent  "C1" -dc {"perm" "measured" 99.4}
      create_element sC1 -type "subpart" -parent  "C1"
        create_fm     "FM003"              -parent "sC1" -dc {"perm" "measured"
79.5}
        create_fm     "FM004"              -parent "sC1" -dc {"perm" "measured"
79.4}
```

- Omit *-fmeda Project_D* key for all subsequent commands.
- Omit *-parent sD1* key for subsequent commands.
- Set parent_prefix so that all subsequent hierarchies can be copied from somewhere else.

The set_scope command does not replace the existing *-fmeda* and *-parent* keys. It sets a default value for those keys to reduce the necessity to duplicate the same entry all over again.

add_parameter

Purpose	Create a new parameter.		
Syntax	add_parameter *parameter_name* -default *value* [-type [<global | *erd_entity* >]] [-description *description*] [-update]		
Arguments	*parameter_name*	Name (identifier) of the parameter.	
	-default *value*	Default value of the parameter.	R
	-type [<global | *erd_entity* >]	The type of the parameter limits its visibility to various commands.	R
	-description *description*	Description of the parameter.	R
	-update	Indicates this command provides additional information for a previous command with the same *parameter_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example based on UB-AB20 and UB-AB21:

```
add_parameter "ATTR_ASIL_LEVEL" -default   d
add_parameter "ATTR_SIL_LEVEL"  -default   4
add_parameter "ASIL_D_CONF"     -default yes
add_parameter "ASIL_D_NO_EFF"   -default 100

create_fmeda "CPU_FMEDA" -type "assumption" -ASIL $ATTR_ASIL_LEVEL  -SIL
$ATTR_SIL_LEVEL

set_scope {"fmeda" "CPU_FMEDA"}
create_element "ALU_X" -type part

set_scope {"parent" "ALU_X"}
create_element "MULT32" -type subpart
create_element "MULT16" -type subpart
create_element "ADD32" -type subpart

set_scope { {"parent" ""} {"parent_prefix" "ALU_X"} }
create_fm "FM_001" -parent "MULT16" -no_effect { {perm $ASIL_D_NO_EFF} {tran
$ASIL_D_NO_EFF} }
create_fm "FM_002" -parent "MULT32" -no_effect { {perm $ASIL_D_NO_EFF} {tran
$ASIL_D_NO_EFF} }
create_fm "FM_003" -parent  "ADD32"
create_fm "FM_004" -parent  "ADD32"
create_fm "FM_005" -parent  "ADD32"

set_scope { {"parent" ""} {"parent_prefix" ""} }

create_sm "SM_001" -class "AoU-SW" -configurable  "no" -dc {{perm 90} {tran 90}}
```

```
{lat 100}}
create_sm "SM_002" -class "HW"     -configurable  "no" -dc {{perm 90} {tran 90}
{lat 100}}
create_sm "SM_003" -class "AoU-HW" -configurable "yes" -dc {{perm 99} {tran 99}
{lat 100}}

assign_sm_fm "SM3_FM1" -sm_name "SM_003" -fm_name "FM_004"  -active $ASIL_D_CONF
```

Usage example based on UB-AB20 and UB-AB21:

```
add_parameter "ATTR_ASIL_LEVEL" -default   d
add_parameter "ATTR_SIL_LEVEL"  -default   4
add_parameter "ASIL_D_CONF"     -default yes
add_parameter "ASIL_D_NO_EFF"   -default 100

load_slf "Project_D.slf"
```

attr_expr

The attr_expr extension provided a support for the conditional usage of given values based on equality or otherwise of a previously declared parameter.

Usage example: Project A has a new parameter “ATTR_ASIL_LEVEL” defined that can take multiple values. A UDA “config” is assigned to the value of the “ATTR_ASIL_LEVEL” parameter. DC values are assigned using the conditional command attr_expr, which allows the use of different DC metrics based on a selected input ASIL level that is passed through the “ATTR_ASIL_LEVEL” parameter. This allows one FMEDA project to store information related to multiple ASIL levels, design configurations, device configurations, and so on within one file without relying on extensions provided by tool vendors.

```
##################
### Project A ####
##################
add_parameter "ATTR_ASIL_LEVEL" -default b
add_attribute "config" -object "create_sm" -default ""

create_sm "SM_001" -class "AoU-SW" -configurable "no" \
           -attribute { "config" $ATTR_ASIL_LEVEL }
           -dc {{perm 90 -attr_expr {config == "d"}} \
                {tran 90 -attr_expr {config == "d"}} \
                {lat 100 -attr_expr {config == "d"}}} \

           -dc {{perm 0 -attr_expr {config == "b"}} \
                {tran 0 -attr_expr {config == "b"}}  \
                {lat 0 -attr_expr {config == "b"}}}


### SoC Project in ASIL D configuration ##############
add_parameter "ATTR_ASIL_LEVEL" -default d
load_slf "Project_A.slf"

### SoC Project in ASIL B configuration ##############
add_parameter "ATTR_ASIL_LEVEL" -default b
load_slf "Project_A.slf"
```

- Expression *attr_expr* looks for user-defined attributes.
- Value of attribute *config* is set to parameter *$ATTR_ASIL_LEVEL*.
- Project A is loaded into SoC-level project with configuration **d**.
- Project A is loaded into SoC-level project with configuration **b**.

assign_fmeda_fmeda

Purpose	Assign fmeda to fmeda.		
Syntax	assign_fmeda_fmeda *fmeda_fmeda_name* -top *top_fmeda_name* -ip { *ip_fmeda_name* } [-attribute { {*name_of_the_user_defined_attribute value*}* }] [-update]		
Arguments	*fmeda_fmeda_name*	Name (identifier) of the assignment.	
	-top *top_fmeda_name*	Name of the top-level FMEDA.	R
	-ip { *ip_fmeda_name* }	List of the FMEDAs to be aggregated into the top_FMEDA.	R
	-attribute { {*name_of_the_user_defined_attribute value*}* }	Sets values of user-defined attributes.	R
	-update	Indicates this command provides additional information for a previous command with the same *fmeda_fmeda_name*.	R
Return value	Returns an empty string if successful, or raises an ERROR if not.		

Usage example:

```
load_slf “Project_A.slf”
load_slf “Project_B.slf”
load_slf “Project_C.slf”

create_fmeda “UC-CB3” -asil “D” -analysis “permanent” -creator “Tier1” -
hierarchical yes
assign_fmeda_fmeda ABC_A -top “UC-CB3” -ip “Project_A”
assign_fmeda_fmeda ABC_B -top “UC-CB3” -ip “Project_B”
assign_fmeda_fmeda ABC_C -top “UC-CB3” -ip “Project_C”
```

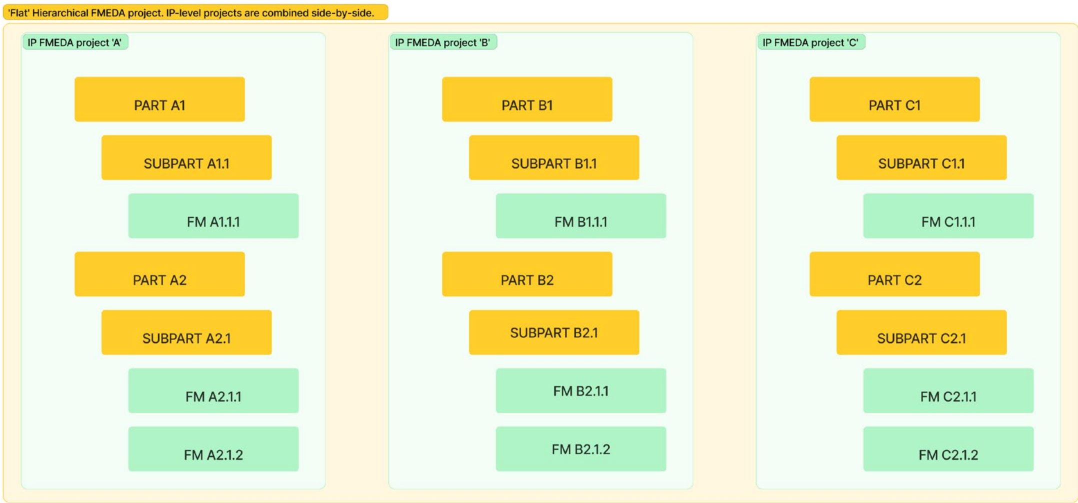


Figure 22. Block diagram of assign_fmeda_fmeda command.

549 assign_fmeda_element

<table>
<tr><td>Purpose</td><td colspan="3">Assign FMEDA project to an element.</td></tr>
<tr><td>Syntax</td><td colspan="3">assign_fmeda_element fmeda_element_name
-mode <summary | detailed>
-target { target_fmeda_name target_safety_hierarchy_object }
-source { source_fmeda_name source_safety_hierarchy_object }
-fmeda fmeda_name
[-description description]
[-update]</td></tr>
<tr><td rowspan="7">Arguments</td><td>fmeda_element_name</td><td>Name (identifier) of the assignment.</td><td></td></tr>
<tr><td>-mode <summary | detailed></td><td>Specifies the how the selected element connects to another FMEDA project.
summary: Converts the existing hierarchy into a “one-liner.” A whole hierarchy and all failure modes should be converted into a top-level hierarchy and an FM using top-level FMEs from the IP. If this is used with -copy = no, then this “one-liner” is recreated each time there is an update trigger action. We also must copy the SM with the AoU class.
detailed: Brings the whole hierarchy from the remote project.</td><td></td></tr>
<tr><td>-target { target_fmeda_name target_safety_hierarchy_object }</td><td>Specifies a project and an element name of the target object. (The target is an object that is being replaced with external information.)</td><td></td></tr>
<tr><td>-source { source_fmeda_name source_safety_hierarchy_object }</td><td>Specifies a project and an element name of the source object.</td><td></td></tr>
<tr><td>-fmeda fmeda_name</td><td>Connects the FS hierarchy to the FMEDA project.</td><td></td></tr>
<tr><td>-description description</td><td>Description of the intended functionality of the Element.</td><td>R</td></tr>
<tr><td>-update</td><td>Indicates this command provides additional information for a previous command with the same fmeda_fmeda_name.</td><td>R</td></tr>
<tr><td>Return value</td><td colspan="3">Returns an empty string if successful, or raises an ERROR if not.</td></tr>
</table>

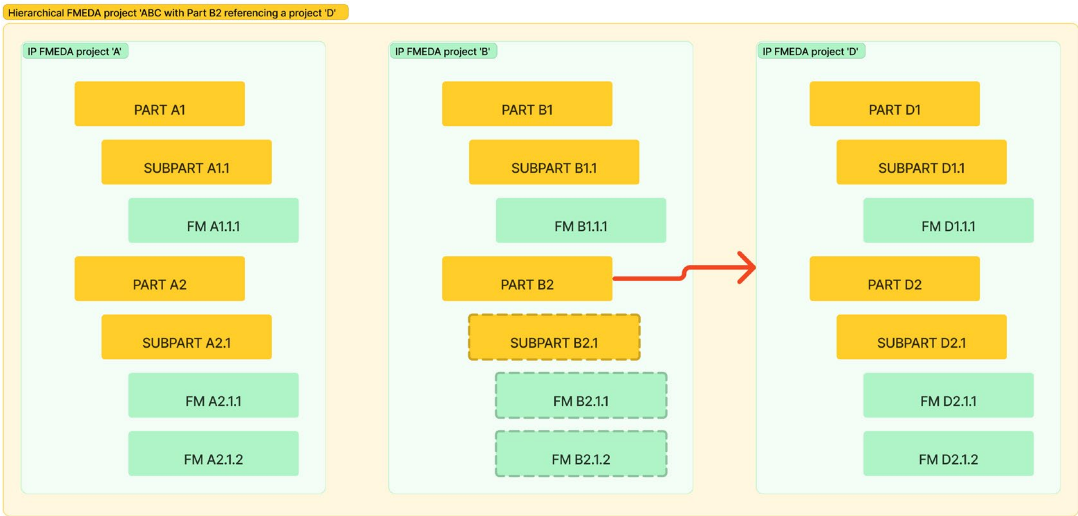


Figure 23. Block diagram of the assign_fmeda_element command.

XI. Annex D – Repository

This document describes validation efforts targeted to validate the data model v0.1 **and** a language at the same time. Given that the language description is far from being finalized, some assumptions were made that are captured in the beginning of this document.

List of additional assumptions:

- SLF - Safety Language Format
- ESLF - Encrypted SLF

A. Example 1

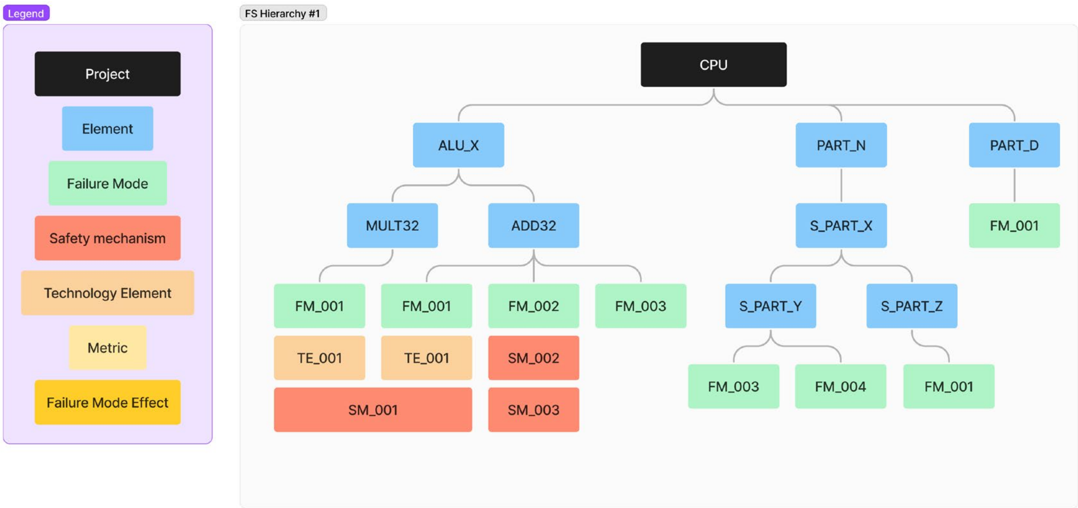


Figure 24. Example

Example of uncompressed code:

```
create_fmeda "CPU_FMEDA" -type "assumption"
  create_element "ALU_X" -type part -fmeda "CPU_FMEDA"
    create_element "ADD32" -type subpart -parent "ALU_X" -fmeda "CPU_FMEDA"
      create_fm "FM_001" -parent "ALU_X.ADD32" -fmeda  "CPU_FMEDA"
      create_fm "FM_002" -parent "ALU_X.ADD32" -fmeda  "CPU_FMEDA"
      create_fm "FM_003" -parent "ALU_X.ADD32" -fmeda  "CPU_FMEDA"
    create_element "MULT32" -type subpart -parent "ALU_X" -fmeda "CPU_FMEDA"
      create_fm "FM_001" -parent "ALU_X.MULT32" -fmeda  "CPU_FMEDA"
  create_element "PARTN" -type part -fmeda "CPU_FMEDA"
    create_element "S_PART_X" -type subpart -parent "PARTN" -fmeda "CPU_FMEDA"
      create_element "S_PART_Z" -type subpart -parent "PARTN.S_PART_X" -fmeda
"CPU_FMEDA"
        create_fm "FM_001" -parent "PARTN.S_PART_X.S_PART_Z" -fmeda  "CPU_FMEDA"
      create_element "S_PART_Y" -type subpart -parent "PARTN.S_PART_X" -fmeda
"CPU_FMEDA"
        create_fm "FM_003" -parent "PARTN.S_PART_X.S_PART_Y" -fmeda  "CPU_FMEDA"
        create_fm "FM_004" -parent "PARTN.S_PART_X.S_PART_Y" -fmeda  "CPU_FMEDA"
  create_element "PARTD" -type part -fmeda "CPU_FMEDA"
    create_fm "FM_001" -parent "PARTD" -fmeda  "CPU_FMEDA"
```

B. Example 2

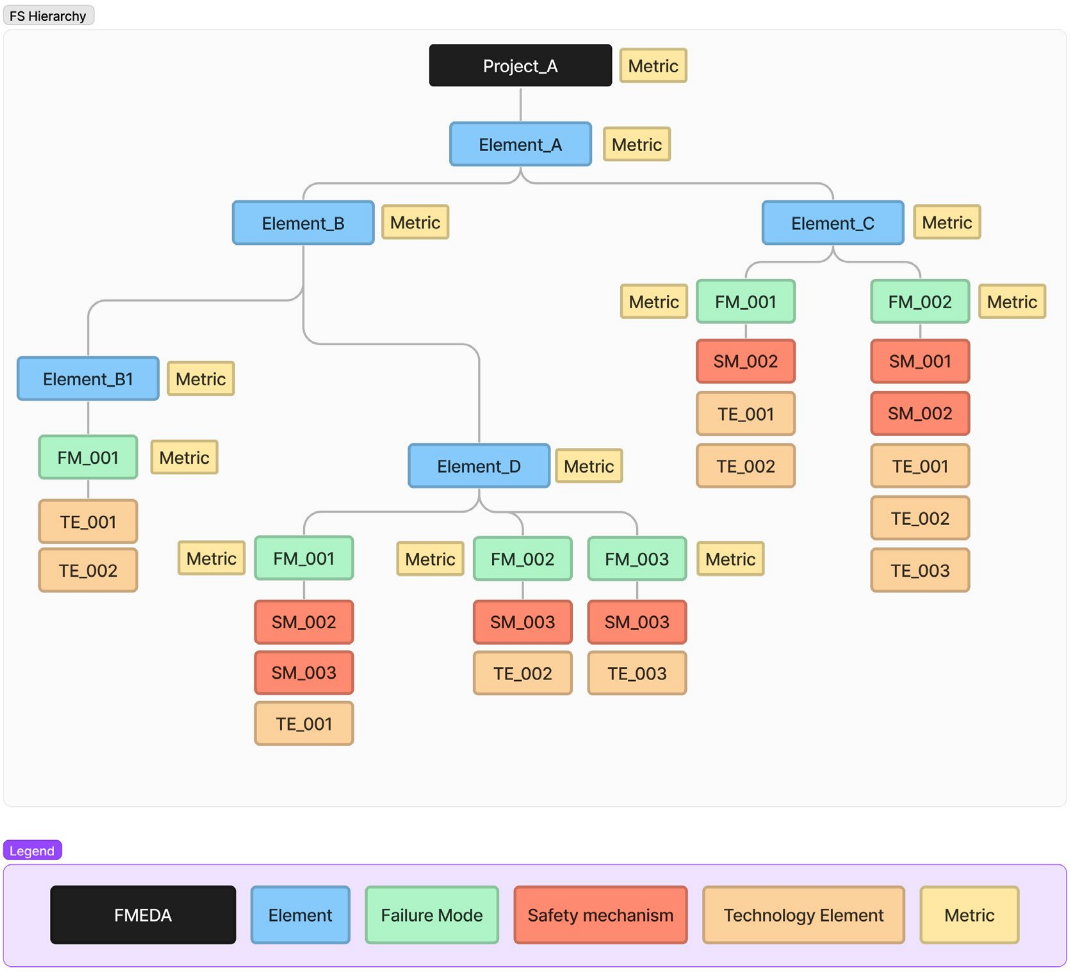


Figure 25. Example

C. Example 3

1. Create an FS hierarchy with multiple levels of subparts.
2. Create an FM for parts and for subparts.
3. Assign multiple technologies to the same FM.
4. Assign sizes to both the FM and the Element; check precedence schema.

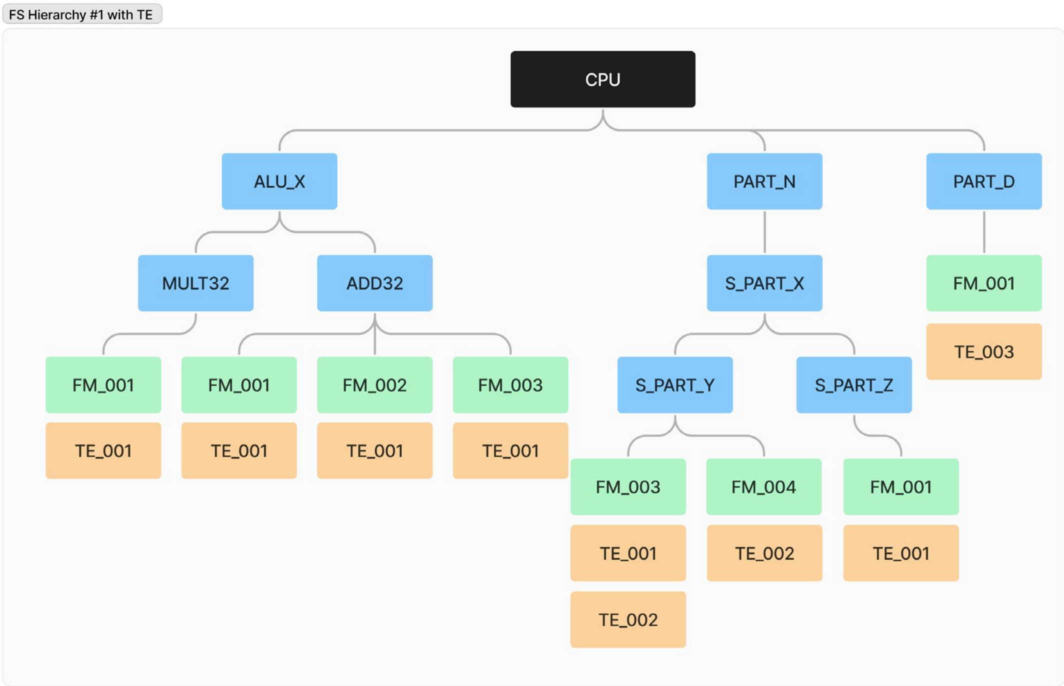


Figure 26. Example

Example of code using only the *element* type of objects:

```
create_fmeda "CPU_FMEDA" -type "assumption"
create_element "ALU_X" -type element -fmeda "CPU_FMEDA"
create_element "ADD32" -type element -parent "ALU_X" -fmeda "CPU_FMEDA"
create_fm "FM_001" -parent "ALU_X.ADD32" -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Digital_5n" -fm_name "FM_001" -parent "ALU_X.ADD32" -
fm_size { absolute perm 15 } -fmeda "CPU_FMEDA"
create_fm "FM_002" -parent "ALU_X.ADD32" -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Digital_5n" -fm_name "FM_002" -parent "ALU_X.ADD32" -
fm_size { absolute perm 5 } -fmeda "CPU_FMEDA"
create_fm "FM_003" -parent "ALU_X.ADD32" -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Digital_5n" -fm_name "FM_003" -parent "ALU_X.ADD32" -
fm_size { absolute perm 10 } -fmeda "CPU_FMEDA"
create_element "MULT32" -type subpart -parent "ALU_X"-fmeda "CPU_FMEDA"
create_fm "FM_001" -parent "ALU_X.MULT32" -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Digital_5n" -fm_name "FM_001" -parent "ALU_X.MULT32"  -
fm_size { absolute perm 35 } -fmeda "CPU_FMEDA"
create_element "PARTN" -type element  -fmeda "CPU_FMEDA"
create_element "S_PART_X" -type element -parent "PARTN" -fmeda "CPU_FMEDA"
assign_te_element -te_name "Analog_5n" -element_name "S_PART_Y" -parent
"PARTN.S_PART_X"  -fm_size { absolute perm 100 }  -fmeda "CPU_FMEDA"
```

```
create_element "S_PART_Z" -type element -parent "PARTN.S_PART_X"  -fmeda
"CPU_FMEDA"
create_fm "FM_001" -parent "PARTN.S_PART_X.S_PART_Z"  -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Digital_5n" -fm_name "FM_001" -parent
"PARTN.S_PART_X.S_PART_Z"  -fm_size { absolute perm 55 } -fmeda "CPU_FMEDA"
create_element "S_PART_Y" -type element -parent "PARTN.S_PART_X"  -fmeda
"CPU_FMEDA"
create_fm "FM_003" -parent "PARTN.S_PART_X.S_PART_Y"  -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Digital_5n" -fm_name "FM_003" -parent
"PARTN.S_PART_X.S_PART_Y" -fm_size { absolute perm 40 } -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Analog_5n"  -fm_name "FM_003" -parent
"PARTN.S_PART_X.S_PART_Y" -fm_size { absolute perm 60 } -fmeda "CPU_FMEDA"
create_fm "FM_004" -parent "PARTN.S_PART_X.S_PART_Y" -fmeda "CPU_FMEDA"
assign_te_fm -te_name "Analog_5n"  -fm_name "FM_004" -parent
"PARTN.S_PART_X.S_PART_Y" -fm_size { absolute perm 20 } -fmeda "CPU_FMEDA"
create_element "PARTD" -type element  -fmeda "CPU_FMEDA"
create_fm "FM_001" -parent "PARTD"  -fmeda "CPU_FMEDA"
assign_te_fm -te_name "RAM_5n" -fm_name "FM_001" -parent "PARTD"  -fm_size {
absolute perm 100 } -fmeda "CPU_FMEDA"
create_te "Analog_5n"  -type "analog"  -fr {perm  3e-9}
create_te "Digital_5n" -type "digital" -fr {perm  1e-9} -fr {tran 8e-9}
create_te "RAM_5n"     -type "ram"     -fr {tran 10e-9}
```

1. Create FMEDA project
2. Create top-level element *ALU_X*
3. Create 2nd-level element *ADD32*, its FMs, link it to TEs
4. Create 2nd-level element *MULT32*, its FMs, link it to TEs
5. Create top-level element *PARTN*
6. Create 2nd-level element *S_PART_Z*, its nested elements, its FMs, link it to TEs
7. Create top-level element `PARTD
8. Create TEs

D. Example 4

Introduction

Review of a simplified block diagram of a safety design of the FIFO module.

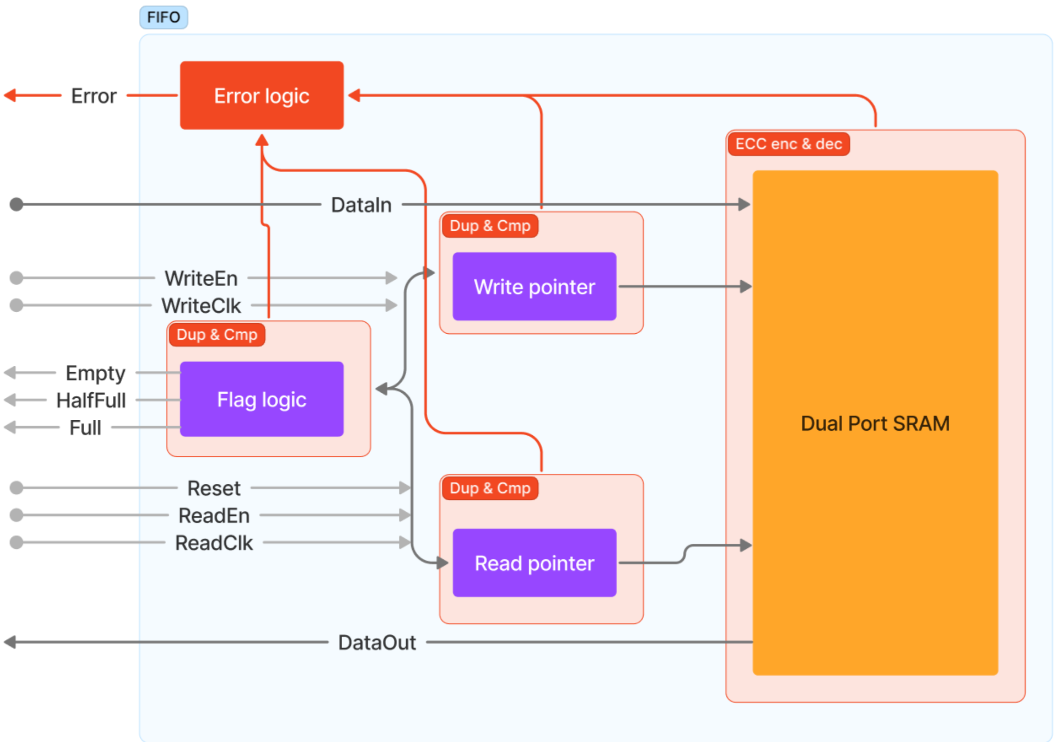


Figure 27. Block diagram of the safety design.

The steps below incrementally present source code of the project as well as diagrams of objects created according to the data model v0.1. A color coding of connections on the diagrams serves only illustrative purposes to ensure a picture with sharp contrast. Please note that connections are also objects of ERD called "relationship" with their own set of attributes. Rectangular boxes are objects of ERD called "object;" an object can have a built-in set of attributes and can also reuse a collection, in which case the object is shown explicitly. Built-in attributes and their values are not shown.

In the illustrations below, the grey box at the right is not a project scope; it is simply a drawing canvas that allows us to logically encapsulate a tool working area for easier review. Data sources that do not expected to exist within a working area of an imaginary software tool will be placed explicitly outside of the drawing canvas. Whether an object belongs to the FMEDA project or not is defined by the existence or absence of a connection net from the leaf object to the FMEDA object on the top, unless otherwise explicitly stated otherwise with a Warning sign below the illustration.

Be aware that this example is an attempt to map existing a simple FMEDA project from a commercial tool to a newly developed draft of a safety language. It highlights the flexibility of a new language that supports all necessary basic constructs and means to store additional metadata.

Step 0. Understand the Difference Between a Language and a Data Model

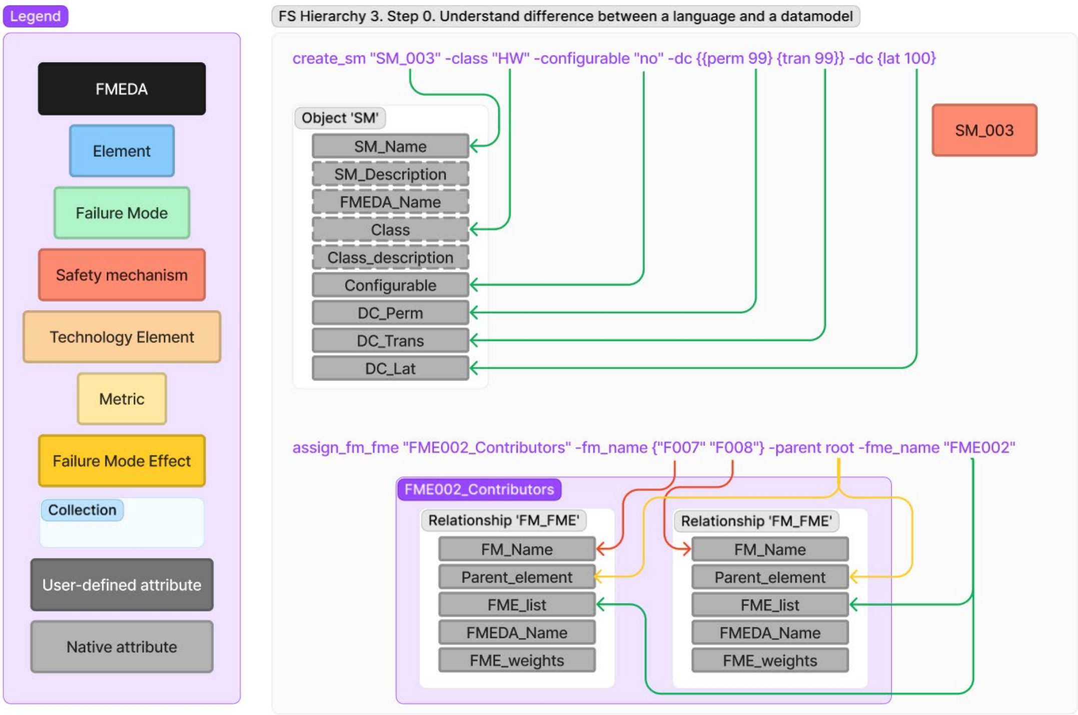


Figure 28. Example of objects created by various commands, and allocation of attributes into predefined data model fields.

Figure 28 demonstrates a crucial difference between a data model and a language. The data model and its requirements define the set of data that must be present in a project regardless of its format (e.g., safety language, Excel file, database). The language defines a way to populate that data in a format that is human-readable and machine-readable. While the Accellera FS WG is using a well-defined framework to ensure consistency of the data model and language, a direct derivation of every command key from every data model's objects' attributes was considered to be extremely wordy. Due to that fact, language commands, while still being directly derived from the data model objects, are more efficient and optimized for the writing of a project manually.

You can see that the *create_sm* command is using multiple ways to store DC metrics in an imaginary object "SM."

The language does not define implementation details of the expected tools' backend. Also, it doesn't define what effect executing a command should be on a tool level. Currently, the language's commands are a data container that hold all necessary data.

Step 1. Create a Library of Collections of Attributes

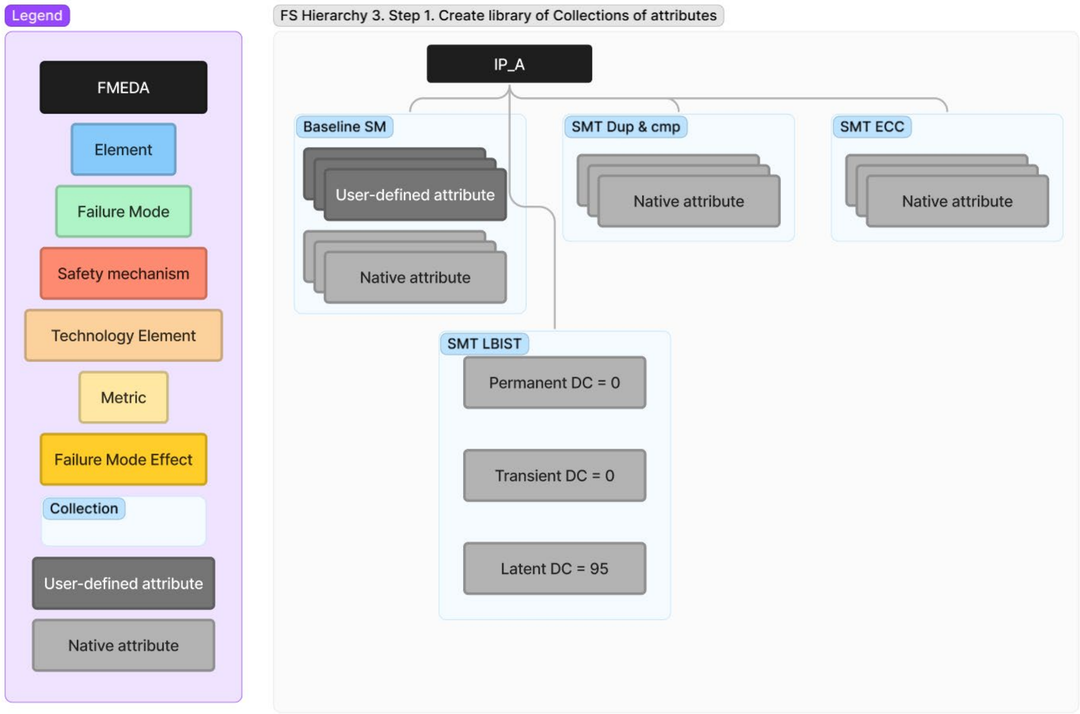


Figure 29. Block diagram of the objects created according to the data model definitions in step 1.

This example is based on data obtained from a commercial tool, and while the set of available fields was purposefully reduced, it’s explicitly shown here that often the user’s intent is to store more data than is supported by baseline objects and their attributes of the data model and language. Nevertheless, the *add_attribute* command enables the storing of metadata in a convenient way. User-defined attributes are a powerful way to store all types of data in the format that is accepted by various tools.

User-defined attributes, while being a standard syntax of the language, cannot be understood equally by all tools. It’s expected that all members of this WG and the broader safety community will communicate back to the WG with proposals of the most prevalent user-defined attributes as baseline attributes for adoption.

A reluctance to contribute back will inevitably cause a fragmentation of an ecosystem and will prevent the correct interoperability of projects.

The second step in enabling reusability within the same project is to use the *add_collection* command as a virtual static container for a set of attributes and their values.

Source code example of creating a project and a library of collections:

```
create_fmeda IP_A -type calculation-based -asil d -sil 4 -analysis all -creator
119lasov -date 10/03/2023 -version 0.1 -data_model_version 0.1 -hierarchical no

# Additional Primary SM attributes
add_attribute "Diagnostic or Avoidance" -object "sm" -default "Avoidance" -type
{enum {"Avoidance" "Diagnostic"}}
add_attribute "Error Response"          -object "sm" -default "HW Error Flag"
add_attribute "ISO26262 DC"             -object "sm" -default "High" -type {enum
{"Low" "Medium" "High"}}
add_attribute "Category"                -object "sm" -default "HW"
add_attribute "Default SM Type"         -object "sm" -default ""
add_attribute "Name"                    -object "sm" -default ""
add_attribute "Primary"                 -object "sm" -default "no" -type {enum
{"yes" "no"}}
add_attribute "Generic comment"         -object "global" -default ""
add_attribute "Equivalent ISO 26262 Diagnostic" -object "sm" -default ""

add_collection "Baseline SM" -object "sm" \
    -list {
        {"Diagnostic or Avoidance" "Diagnostic"} \
        {"Error Response" "HW Error Flag"} \
        {"ISO26262 DC" "High"} \
        {"Category" "HW"} \
        {"configurable" "no"} \
        {"class" "HW"} \
        {"fmeda" "IP_A"} \
    }
    -fmeda IP_A

# Each collection represents one Safety Mechanism Type
add_collection "SMT Dup & cmp" -object "sm" -list { {"dc" {perm 95}} {"dc" {tran
90}} {"dc" {lat 0}} } -fmeda IP_A
add_collection "SMT ECC"       -object "sm" -list { {"dc" {perm 99}} {"dc" {tran
99}} {"dc" {lat 0}} } -fmeda IP_A
add_collection "SMT LBIST"     -object "sm" -list { {"dc" {perm 0}} {"dc" {tran
0}} {"dc" {lat 95}} } -fmeda IP_A
```

Step 2. Create a Library of Safety Mechanisms

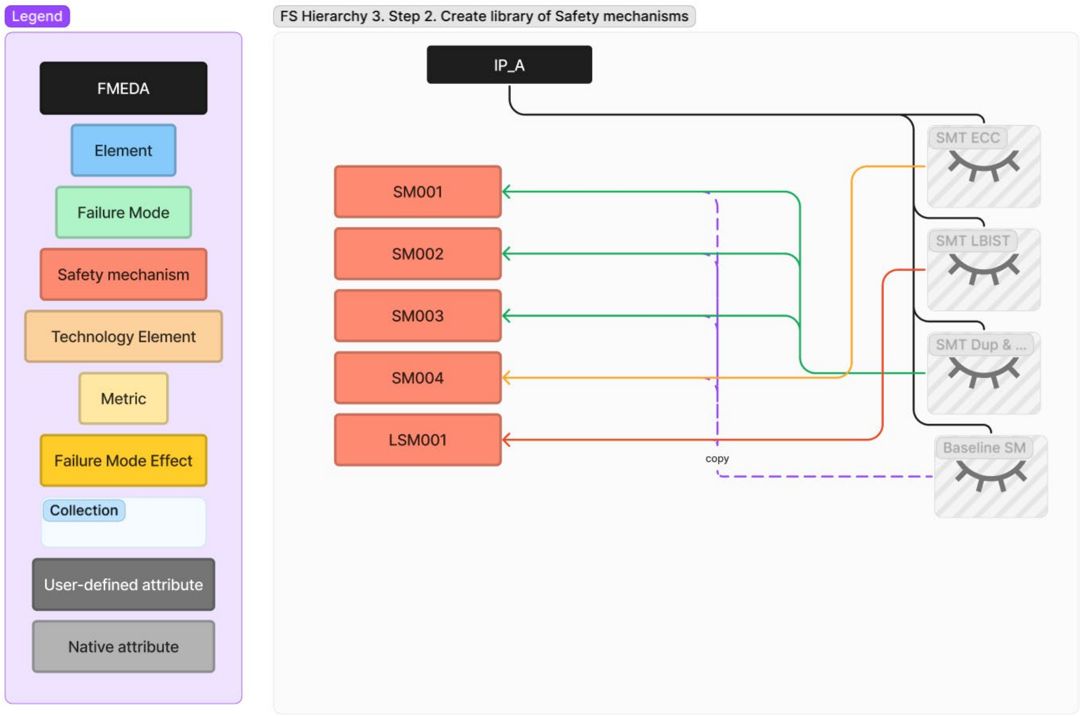


Figure 30. Block diagram of the objects created according to the data model definitions in step 2.

While the language supports Safety Mechanisms as objects attached to an FMEDA project, it also supports Safety Mechanisms independently. In this particular example, there's no specific goal in having Safety Mechanisms not connected to the FMEDA project, but it is more closely aligned with the user's intent.

Source code example of creating a library of Safety mechanisms:

```
create_sm "SM001" -collection "Baseline SM" -collection "SMT Dup & cmp" \
    -attribute {{"Primary" "no"} {"Name" "Flag Logic Dup"}} \
    -attribute {"Equivalent ISO 26262 Diagnostic" "Processing units:
Regis-ters::HW redundancy (e.g.dual core lockstep, asym-metric redundancy, coded
processing)"}

create_sm "SM002" -collection "Baseline SM" -collection "SMT Dup & cmp" \
    -attribute {{"Primary" "no"} {"Name" "WR Logic Dup"}}
    -attribute {"Equivalent ISO 26262 Diagnostic" "Processing units:
Regis-ters::HW redundancy (e.g.dual core lockstep, asym-metric redundancy, coded
processing)"}

create_sm "SM003" -collection "Baseline SM" -collection "SMT Dup & cmp" \
    -attribute {{"Primary" "no"} {"Name" "RD Logic Dup"}} \
    -attribute {"Equivalent ISO 26262 Diagnostic" "Processing units:
Regis-ters::HW redundancy (e.g.dual core lockstep, asym-metric redundancy, coded
processing)"}

create_sm "SM004" -collection "Baseline SM" -collection "SMT ECC" \
    -attribute {{"Primary" "no"} {"Name" "ECC"}} \
```

```
    -attribute {"Equivalent ISO 26262 Diagnostic" "Volatile memory::Memory
monitoring using error-detection-correction codes(EDC)"}

create_sm "LSM001" -collection "Baseline SM" \
    -collection "SMT LBIST" \
    -attribute {
        {"Primary" "no"} \
        {"Name" "LBIST"} \
        {"Error Response" "Abort"} \
        {"ISO26262 DC" "Medium"} \
        {"Equivalent ISO 26262 Diagnostic" "Processing units: Registers::Self-test
supported by hardware(one-channel)"}
    }
```

Step 3. Create the Safety Hierarchy

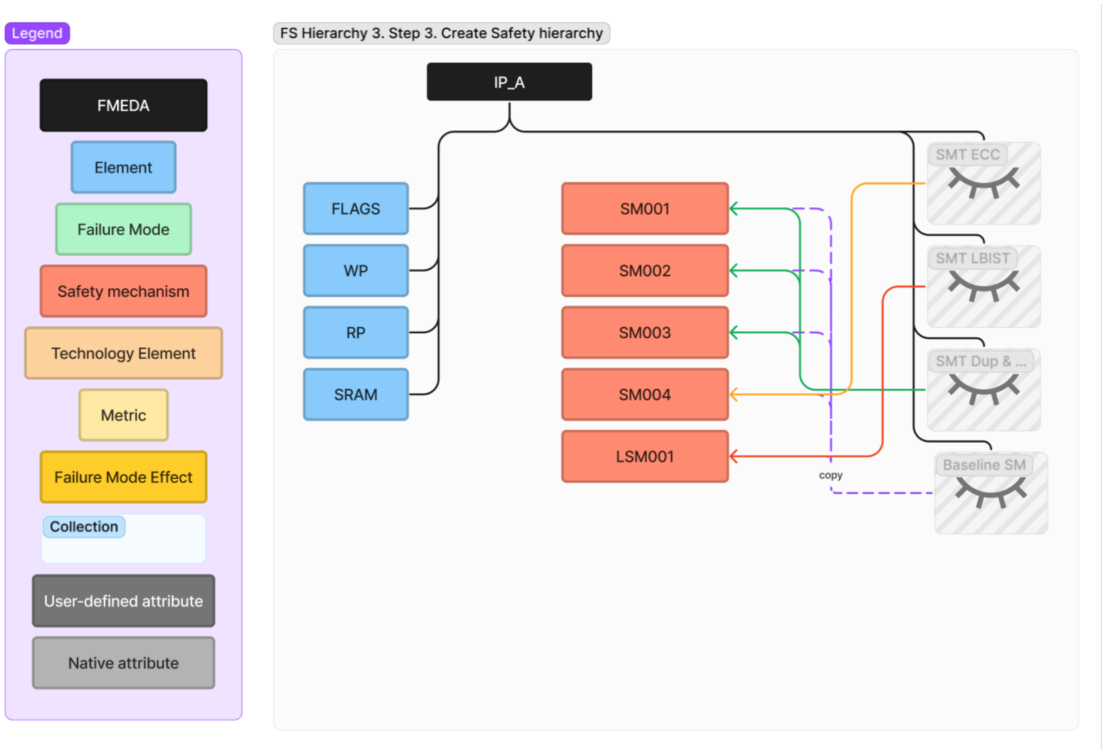


Figure 31. Block diagram of the objects created according to the data model definitions in step 3.

The FIFO demo project is quite simple, so here we have created four parts.

Source code example of creating a Safety hierarchy:

```
create_element FLAGS -parent IP_A -type part -fmeda IP_A -description "Status
flags control logic"
create_element WP    -parent IP_A -type part -fmeda IP_A -description "Write
pointer logic"
create_element RP    -parent IP_A -type part -fmeda IP_A -description "Read
pointer logic"
create_element SRAM  -parent IP_A -type part -fmeda IP_A -description "SRAM
memory"
```

Step 4. Create Failure Modes and Assisting Collections

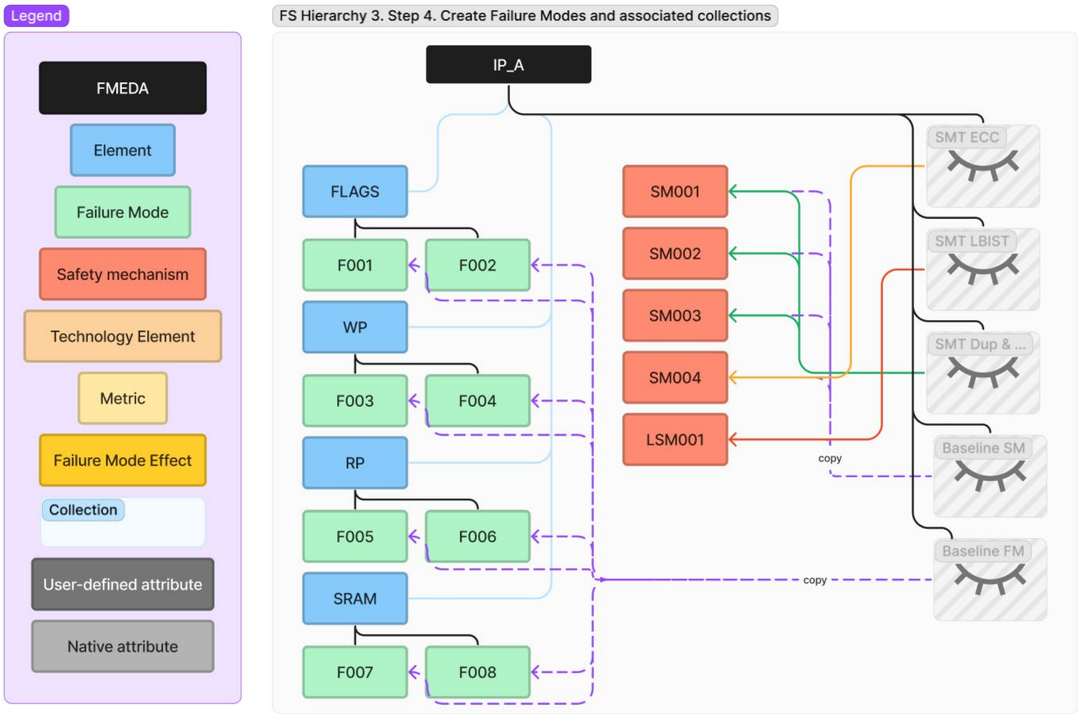


Figure 32. Block diagram of the objects created according to the data model definitions in step 4.

For each part we create two Failure modes. One is to be associated later with the actual design hierarchy, and the other one with the safety mechanism hierarchy. This difference is reflected by using various values for the os *-type* attribute.

Source code example of creating Failure modes and assisting collections:

```
add_attribute "Probability to violate Safety Goal"        -object "fm" -default
"no" -type {enum {"yes" "no"}}
add_attribute "Systematic or random failure"              -object "fm" -default
"random" -type {enum {"systematic" "random"}}
add_attribute "Potential faults"                          -object "fm" -default ""
add_attribute "Potential errors"                          -object "fm" -default ""
add_attribute "Permanent ot transient"                    -object "fm" -default ""
-type {enum {"permanent" "transient"}}
add_attribute "Potential Cause of SM Fault"               -object "fm" -default ""
-type {enum {"SEU" "TDDB"}}
add_attribute "ISO 26262 Equivalent Fault/Error/Failure" -object "fm" -default ""


add_collection "Baseline FM" -object "fm" \
    -list {
        {"Probability to violate Safety Goal" "yes"} \
        {"Systematic or random failure" "random"} \
        {"safety_relevant" "yes"}
    }
    -fmeda IP_A
```

```
create_fm "F001" -parent FLAGS -type mission -safety_relevant yes -dc_aggregation
max \
    -no_effect { {perm 0} {tran 100} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "Flag logic is faulty"} \
                {"Potential errors"            "Incorrect flag indication"} \
                {"Potential Cause of SM Fault" "TDDB"} \
                {"ISO 26262 Equivalent Fault/Error/Failure" "Processing units:
Registers::Stackoverflow/underflow"}}
    -collection "Baseline FM"

create_fm "F002" -parent FLAGS -type passive -safety_relevant yes -dc_aggregation
max \
    -no_effect { {perm 100} {tran 0} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "Flag logic is faulty"} \
                {"Potential errors"            "Incorrect flag indication"} \
                {"Potential Cause of SM Fault" "SEU"} \
                {"ISO 26262 Equivalent Fault/Error/Failure" "Processing units:
Registers::Stackoverflow/underflow"}}
    -collection "Baseline FM"

create_fm "F003" -parent WP -type mission -safety_relevant yes -dc_aggregation max
\
    -no_effect { {perm 0} {tran 100} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "WR logic is faulty"} \
                {"Potential errors"            "Incorrect WR ptr to SRAM"} \
                {"Potential Cause of SM Fault" "TDDB"}
                {"ISO 26262 Equivalent Fault/Error/Failure" "Processing units:
Registers::Stackoverflow/underflow"}}
    -collection "Baseline FM"

create_fm "F004" -parent WP -type passive -safety_relevant yes -dc_aggregation max
\
    -no_effect { {perm 100} {tran 0} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "WR logic is faulty"} \
                {"Potential errors"            "Incorrect WR ptr to SRAM"} \
                {"Potential Cause of SM Fault" "SEU"} \
                {"ISO 26262 Equivalent Fault/Error/Failure" "Processing units:
Registers::Stackoverflow/underflow"}}
    -collection "Baseline FM"

create_fm "F005" -parent RP -type mission -safety_relevant yes -dc_aggregation max
\
    -no_effect { {perm 0} {tran 100} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "RP logic is faulty"} \
                {"Potential errors"            "Incorrect RD ptr to SRAM"} \
                {"Potential Cause of SM Fault" "TDDB"} \
                {"ISO 26262 Equivalent Fault/Error/Failure" "Processing units:
Registers::Stackoverflow/underflow"}}
    -collection "Baseline FM"

create_fm "F006" -parent RP -type passive -safety_relevant yes -dc_aggregation max
\
    -no_effect { {perm 100} {tran 0} } \
    -perceived { {perm 0} {tran 0} } \
```

```
    -attribute {{"Potential faults"            "RP logic is faulty"} \
               {"Potential errors"            "Incorrect RD ptr to SRAM"} \
               {"Potential Cause of SM Fault" "SEU"} \
               {"ISO 26262 Equivalent Fault/Error/Failure" "Processing units:
Registers::Stackoverflow/underflow"}}
    -collection "Baseline FM"

create_fm "F007" -parent SRAM -type mission -safety_relevant yes -dc_aggregation
max \
    -no_effect { {perm 0} {tran 100} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "Failure in SRAM bits"} \
               {"Potential errors"            "Corrupted data in SRAM"} \
               {"Potential Cause of SM Fault" "TDDB"} \
               {"ISO 26262 Equivalent Fault/Error/Failure" "Volatile memory::d.c.
faults model(addr,data,control)"}}
    -collection "Baseline FM"

create_fm "F008" -parent SRAM -type active -safety_relevant yes -dc_aggregation
max \
    -no_effect { {perm 100} {tran 0} } \
    -perceived { {perm 0} {tran 0} } \
    -attribute {{"Potential faults"            "Failure in SRAM bits"} \
               {"Potential errors"            "Corrupted data in SRAM"} \
               {"Potential Cause of SM Fault" "SEU"} \
               {"ISO 26262 Equivalent Fault/Error/Failure" "Volatile memory::d.c.
faults model(addr,data,control)"}}
    -collection "Baseline FM"
```

Step 5. Assign Safety Mechanisms to Failure Modes

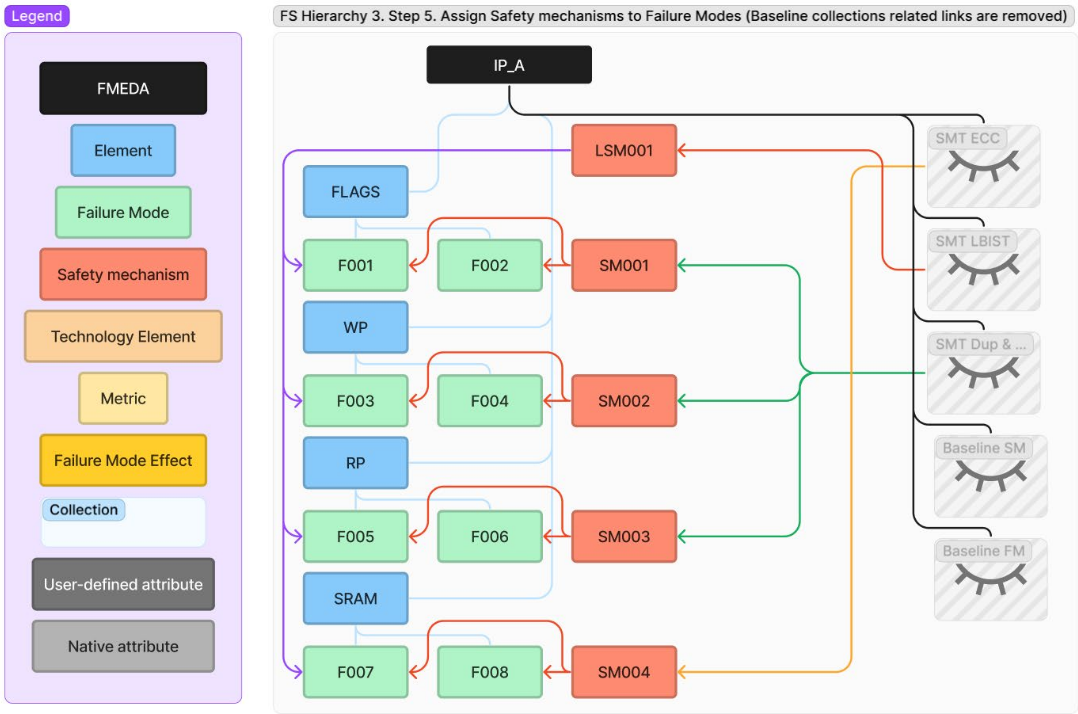


Figure 33. Block diagram of the objects created according to the data model definitions in step 5.

Links related to baseline collections have been removed from the image.

Source code example of creating a link from Safety mechanisms to Failure modes:

```
assign_sm_fm PSM_to_perm_FM_FLAG -sm_name SM001 -fm_name {"F001" "F002"} -parent
IP_A -fmeda IP_A -active yes
assign_sm_fm PSM_to_perm_FM_WP   -sm_name SM002 -fm_name {"F003" "F004"} -parent
IP_A -fmeda IP_A -active yes
assign_sm_fm PSM_to_perm_FM_RP   -sm_name SM003 -fm_name {"F005" "F006"} -parent
IP_A -fmeda IP_A -active yes
assign_sm_fm PSM_to_perm_FM_SRAM -sm_name SM004 -fm_name {"F007" "F008"} -parent
IP_A -fmeda IP_A -active yes
assign_sm_fm LBSIT_to_all_latent_FM -sm_name LSM001 -fm_name {"F001" "F003" "F005"
"F007"} -parent IP_A -fmeda IP_A -active yes
```

Step 6. Create Technology Elements

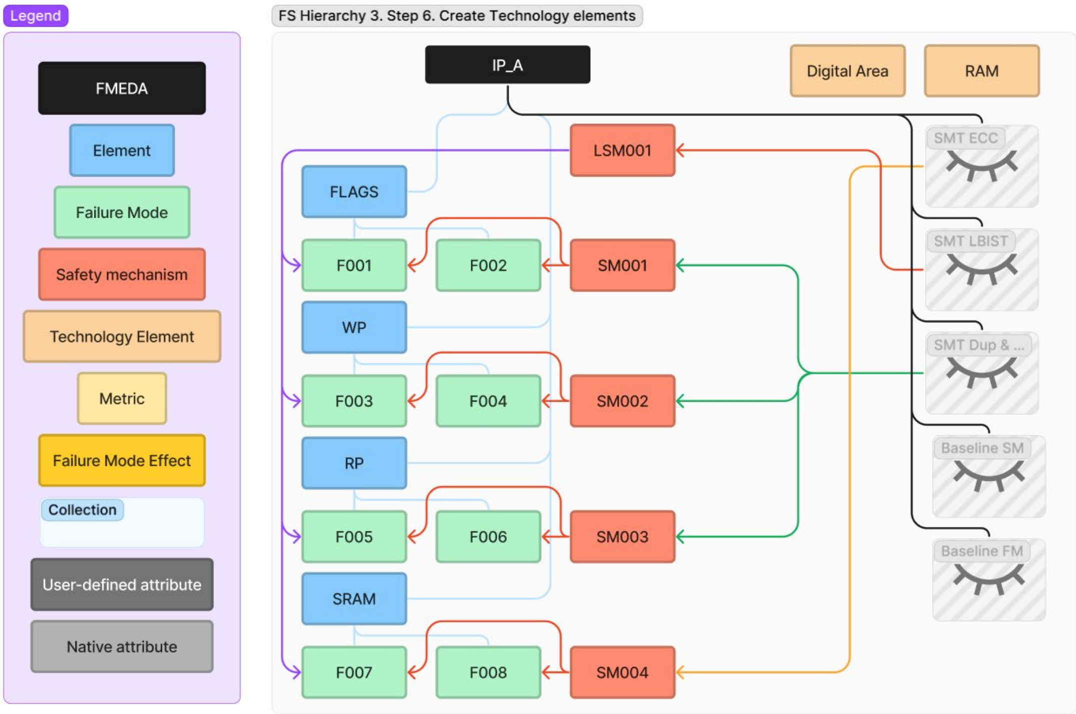
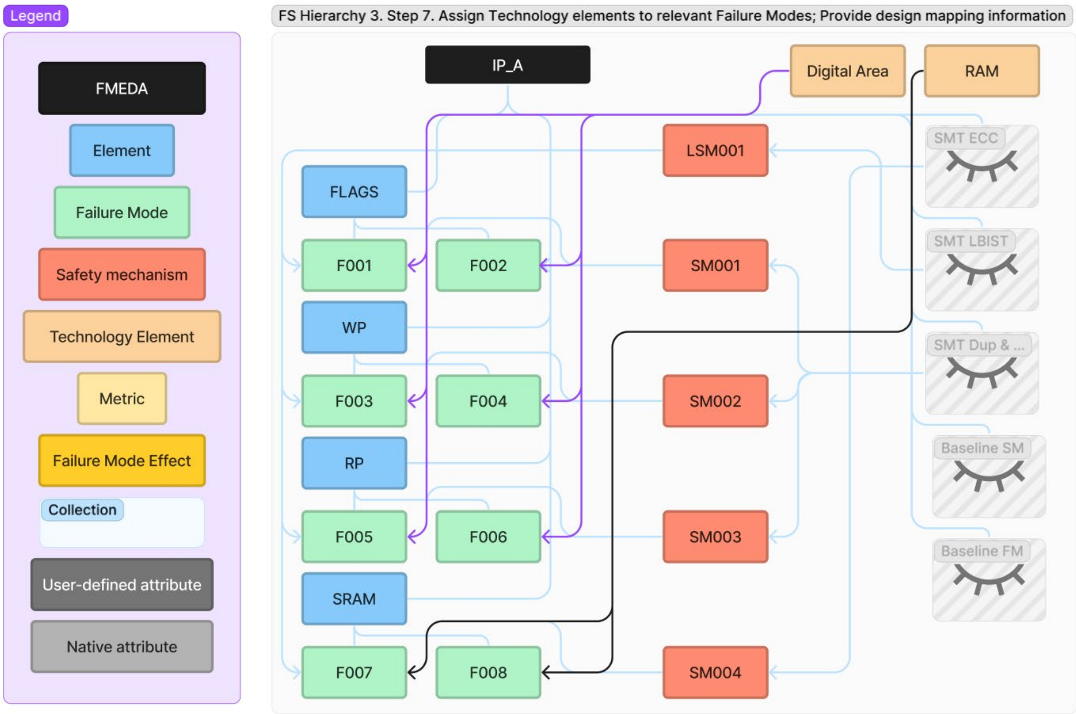


Figure 34. Block diagram of the objects created according to the data model definitions in step 6.

Source code example of creating technology elements:

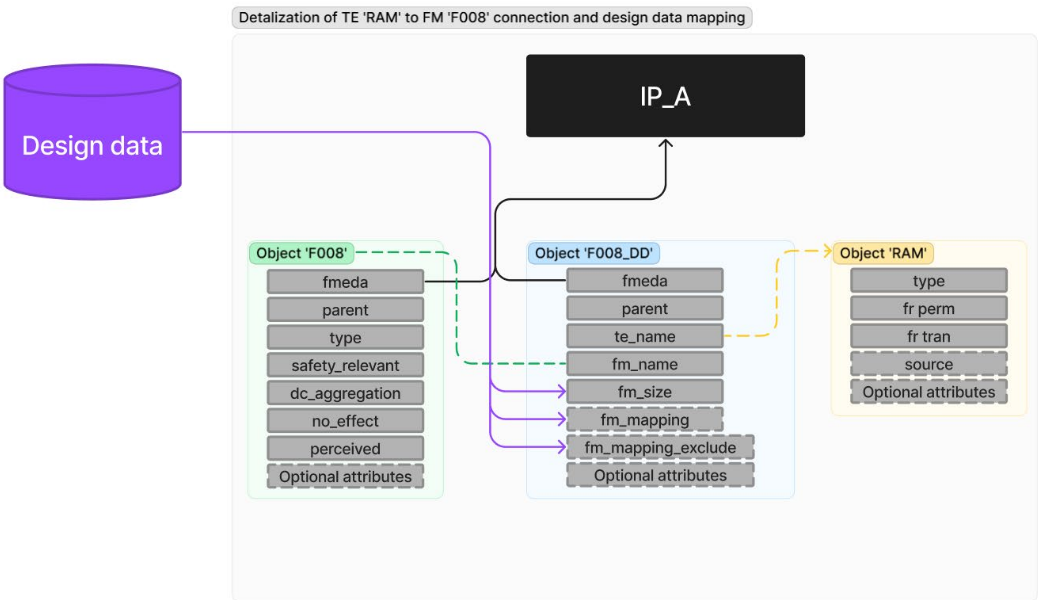
```
create_te "Digital_Area" -type digital -source IEC_62380 -fr {{perm 0.03033} {tran
0}      }
create_te "Analog_Area"  -type analog  -source IEC_62380 -fr {{perm 0.03033} {tran
0.01}   }
create_te "ROM"          -type ram     -source IEC_62380 -fr {{perm 0.03033} {tran
1e-7}   }
create_te "RAM"          -type rom     -source IEC_62380 -fr {{perm 0.03033} {tran
1e-7}   }
create_te "Flops"        -type digital -source IEC_62380 -fr {{perm 0}       {tran
3.4e-6}}
```

939 ## Step 7. Assign Technology Elements to Failure Modes, Mapping



940

941 *Figure 35. Block diagram of the objects created according to the data model definitions in step 7.*



942

943 *Figure 36. Detalization of a TE-to-FM connection with design data mapping. Native attributes are shown to*
944 *illustrate internal data structures.*

945

Source code example of creating a link from Technology elements to Failure modes with design data mapping:

```
add_attribute "TE_Gates" -object "te" -default 0
add_attribute "TE_Flops" -object "te" -default 0
assign_te_fm "F001_DD" -te_name "Digital_Area" -fm_name "F001" -parent IP_A -fmeda
IP_A -fm_size {absolute perm 330.85} -fm_mapping {"test.DUT.FL_IF"} -attribute
{{"TE_Gates" 34} {"TE_Flops" 4}}
assign_te_fm "F002_DD" -te_name "Digital_Area" -fm_name "F002" -parent IP_A -fmeda
IP_A -fm_size {absolute perm 330.85} -fm_mapping {"test.DUT.FL_SM"} -attribute
{{"TE_Gates" 34} {"TE_Flops" 4}}
assign_te_fm "F003_DD" -te_name "Digital_Area" -fm_name "F003" -parent IP_A -fmeda
IP_A -fm_size {absolute perm 147.46} -fm_mapping {"test.DUT.RP_IF"} -attribute
{{"TE_Gates" 8} {"TE_Flops" 3}}
assign_te_fm "F004_DD" -te_name "Digital_Area" -fm_name "F004" -parent IP_A -fmeda
IP_A -fm_size {absolute perm 147.46} -fm_mapping {"test.DUT.RP_SM"} -attribute
{{"TE_Gates" 8} {"TE_Flops" 3}}
assign_te_fm "F005_DD" -te_name "Digital_Area" -fm_name "F005" -parent IP_A -fmeda
IP_A -fm_size {absolute perm 158.52} -fm_mapping {"test.DUT.WP_IF"} -attribute
{{"TE_Gates" 10} {"TE_Flops" 3}}
assign_te_fm "F006_DD" -te_name "Digital_Area" -fm_name "F006" -parent IP_A -fmeda
IP_A -fm_size {absolute perm 158.52} -fm_mapping {"test.DUT.WP_SM"} -attribute
{{"TE_Gates" 10} {"TE_Flops" 3}}
assign_te_fm "F007_DD" -te_name "RAM" -fm_name "F007" -parent IP_A -fmeda IP_A -
fm_size {absolute bits 192.00} -fm_mapping {"test.DUT.sdpram_i1.sdpram_i1"} -
attribute {{"TE_Gates" 2} {"TE_Flops" 0}}
assign_te_fm "F008_DD" -te_name "RAM" -fm_name "F008" -parent IP_A -fmeda IP_A -
fm_size {absolute bits 192.00} -fm_mapping {"test.DUT.sdpram_i1.sdpram_i1"} -
attribute {{"TE_Gates" 2} {"TE_Flops" 0}}
```

Step 8. Create Failure Mode Effects and Connect them to Failure Modes

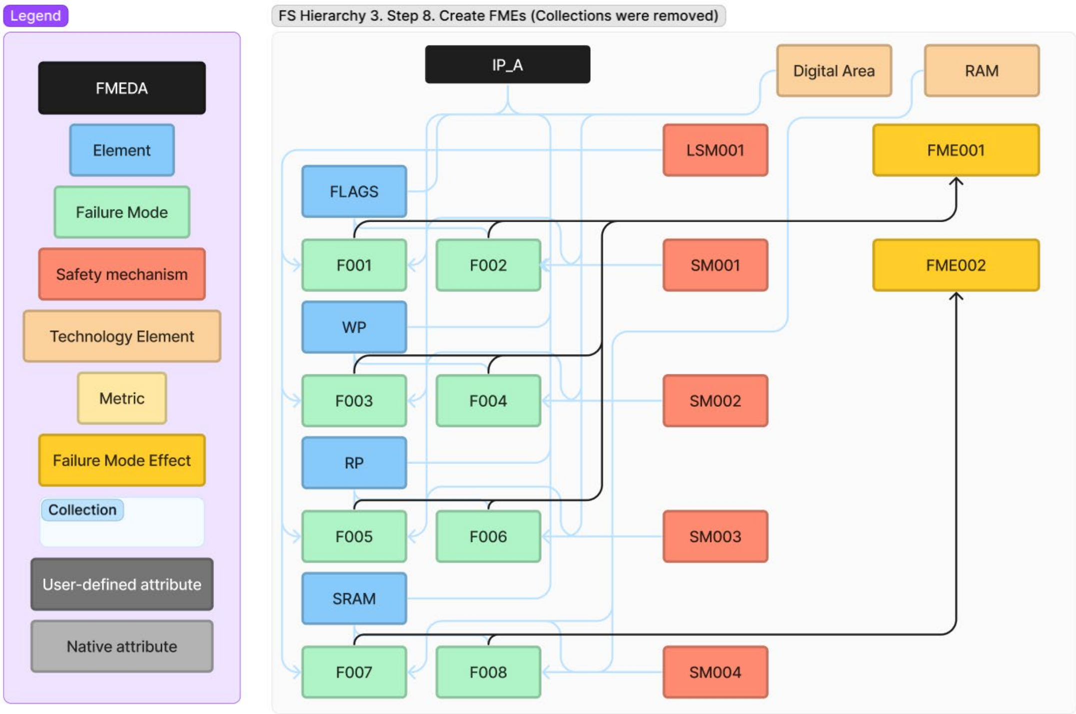


Figure 37. Block diagram of the objects created according to the data model definitions in step 8.

Source code example of creating a link from Technology elements to Failure modes, mapping.

```
create_fme "FME001" -fmeda IP_A -description "Loss of data"
create_fme "FME002" -fmeda IP_A -description "Incorrect data"
assign_fm_fme "FME001_Contributors" -fmeda IP_A -fm_name {"F001" "F002" "F003"
"F004" "F005" "F006"} -parent IP_A -fme_name "FME001" -fme_weight {1 1 1 1 1 1}
assign_fm_fme "FME002_Contributors" -fmeda IP_A -fm_name {"F007" "F008"} -parent
IP_A -fme_name "FME002" -fme_weight {1 1}
```

Step 9. Update Objects According to Verification Strategy

It is assumed here that an integrated toolchain is used to connect FMEDA data to verification data. The bare minimum subset of the data to be shared is observation and detection points, mapping to a design hierarchy. Fault simulation settings are not reflected in this example, although for traceability purposes we need to have that connection.

Nevertheless, as of today a fault campaign object is not considered to be a part of an FMEDA analysis, and there is no construct that would allow a user to create a new type of object. Therefore, pointers to the verification data can be stored as user-defined attributes of the *create_fmeda* command, thus enabling a baseline traceability from measured metrics back to fault simulation results. Such usage is not shown in this example, however.

Please note the use of the *-update* key to update already created objects.

Source code example of updating Failure modes with verification information:

```
add_attribute "Observation points" -object "fm" -default ""
create_fm "F001" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
"test.DUT.FL_IF.Empty_ test.DUT.FL_IF.Full_ test.DUT.FL_IF.HalfFull_"}
create_fm "F002" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
""}
create_fm "F003" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
"test.DUT.WP_IF.Count"}
create_fm "F004" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
""}
create_fm "F005" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
"test.DUT.RP_IF.Count"}
create_fm "F006" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
""}
create_fm "F007" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
"test.DUT.sdpram_i1.sdpram_i1.L_DataOut test.DUT.sdpram_i1.sdpram_i1.R_DataOut"}
create_fm "F008" -update -parent IP_A -fmeda IP_A -attribute {"Observation points"
""}
```

Source code example of updating Safety mechanisms with verification information:

```
add_attribute "Diagnostic points" -object "sm" -default ""
create_sm "SM001" -update -attribute {"Diagnostic points" "test.DUT.FlagError"}
create_sm "SM002" -update -attribute {"Diagnostic points" "test.DUT.WriteError"}
create_sm "SM003" -update -attribute {"Diagnostic points" "test.DUT.ReadError"}
create_sm "SM004" -update -attribute {"Diagnostic points"
"test.DUT.sdpram_i1.EccError"}
```

Source code example of updating Failure modes with results of a digital fault simulation campaign:

```
create_fm "F001" -update -parent IP_A -fmeda IP_A -dc {perm measured 95.45}
create_fm "F003" -update -parent IP_A -fmeda IP_A -dc {perm measured 94.44}
create_fm "F005" -update -parent IP_A -fmeda IP_A -dc {perm measured 94.44}
create_fm "F007" -update -parent IP_A -fmeda IP_A -dc {perm measured 40.14} -
no_effect {perm 16.47}
```

Step 10. Create FMEDA-scoped Metrics

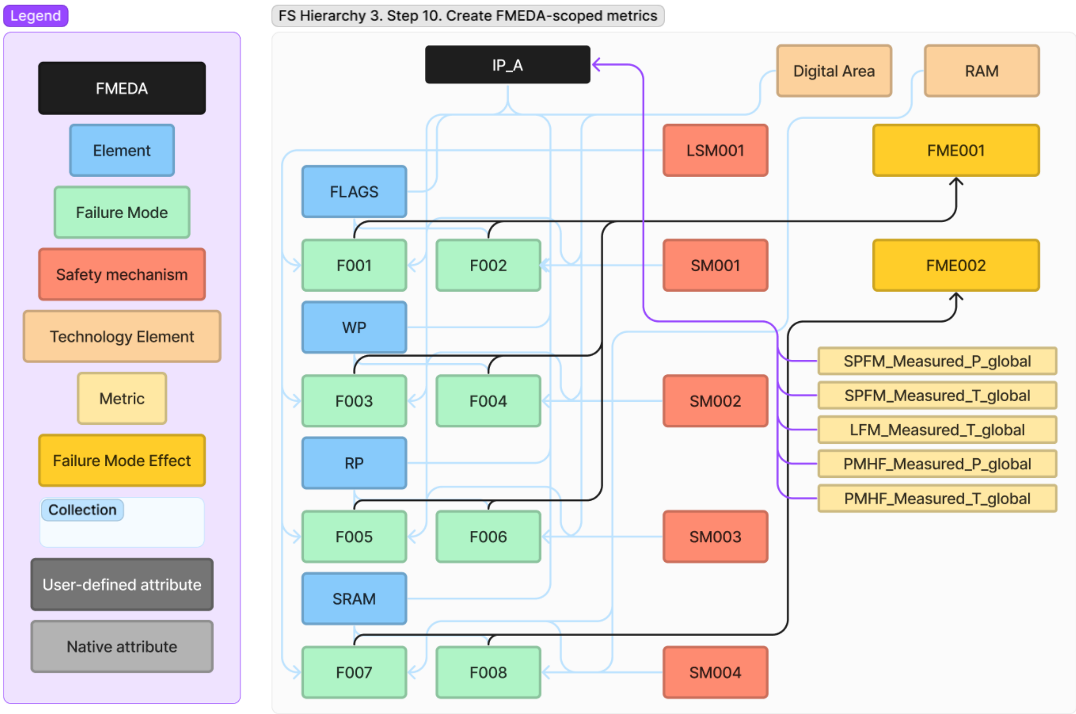


Figure 38. Block diagram of the objects created according to the data model definitions in step 10.

Source code example of creating metrics:

```
add_collection "Baseline FMEDA metric" -object "metric" \
    -list {
        {"scope" {fmeda IP_A}} \
        {"te_name" {"Digital_Area" "RAM"}} \
        {"fmeda" "IP_A"}
    }
    -fmeda IP_A

define_metric_iso26262 SPFM_Measured_P_global -metric_type {spfm 91.96} -
analysis_type perm -collection "Baseline FMEDA metric"
define_metric_iso26262 SPFM_Measured_T_global -metric_type {spfm 97.95} -
analysis_type tran -collection "Baseline FMEDA metric"
define_metric_iso26262 LFM_Measured_T_global  -metric_type {lfm 92.74}  -
analysis_type perm -collection "Baseline FMEDA metric"
define_metric_iso26262 PMHF_Measured_P_global -metric_type {pmhf 4.970} -
analysis_type perm -collection "Baseline FMEDA metric"
define_metric_iso26262 PMHF_Measured_T_global -metric_type {pmhf 1.786E-6} -
analysis_type tran -collection "Baseline FMEDA metric"
```

Step 11. Create FME-scoped Metrics

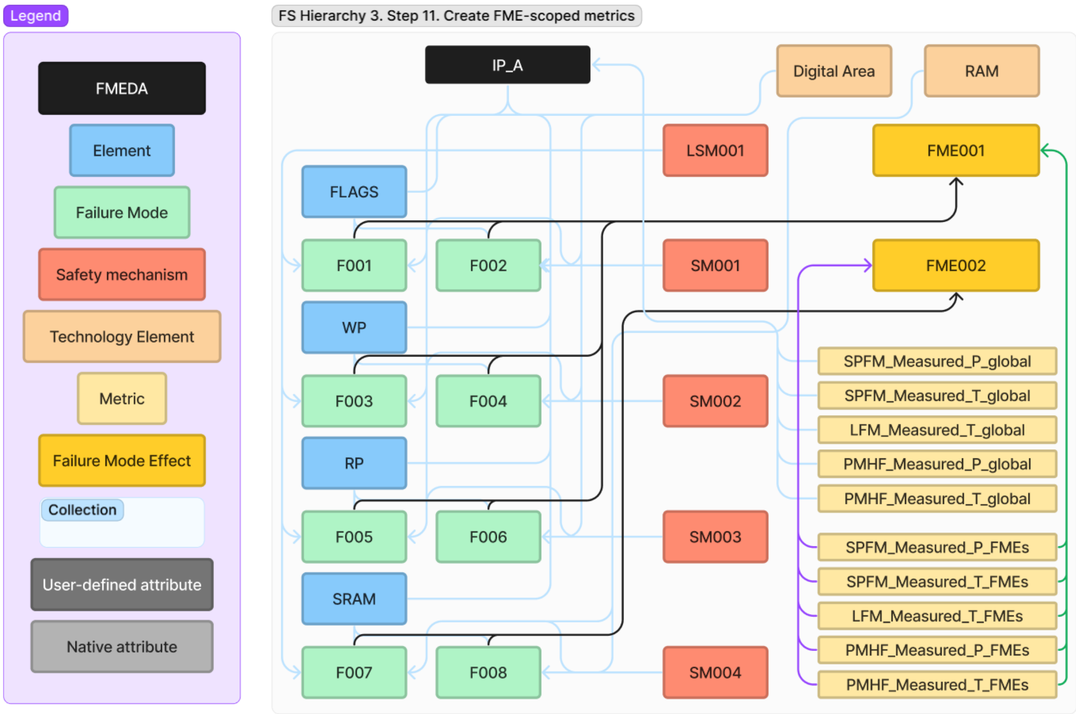


Figure 39. Block diagram of the objects created according to the data model definitions in step 11.

In this case, FMEDA-scoped metrics are no different from FME-scoped metrics due to the way that FMEs are connected to FMs.

Source code example of creating metrics:

```
add_collection "Baseline FME metric" -object "metric" \
    -list {
        {"scope" {fme "FME001" "FME002"}} \
        {"te_name" {"Digital_Area" "RAM"}} \
        {"fmeda" "IP_A"}
    }
    -fmeda IP_A

define_metric_iso26262 SPFM_Measured_P_FMEs -metric_type {spfm 91.96} -
analysis_type perm -collection "Baseline FME metric"
define_metric_iso26262 SPFM_Measured_T_FMEs -metric_type {spfm 97.95} -
analysis_type tran -collection "Baseline FME metric"
define_metric_iso26262 LFM_Measured_T_FMEs  -metric_type {lfm 92.74}  -
analysis_type perm -collection "Baseline FME metric"
define_metric_iso26262 PMHF_Measured_P_FMEs -metric_type {pmhf 4.970} -
analysis_type perm -collection "Baseline FME metric"
define_metric_iso26262 PMHF_Measured_T_FMEs -metric_type {pmhf 1.786E-6} -
analysis_type tran -collection "Baseline FME metric"
```

Data Tracing

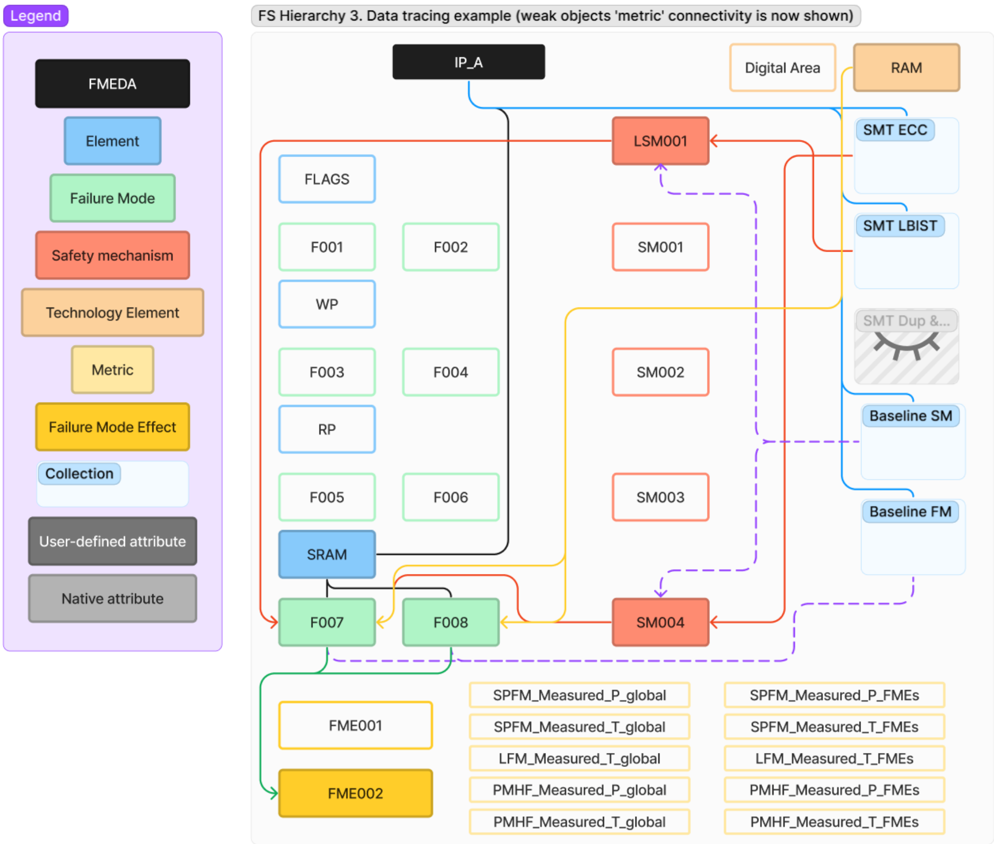


Figure 40. Block diagram of the objects created according to the data model definitions. Only objects related to element SRAM are highlighted.

Figure 40 shows how data tracing can be done using the data model. The operation on a dataset—as on a set of interlinked objects—enables very detailed introspection capabilities. As of today, the language does not support introspection capabilities or any kind of queries to internal objects. Nevertheless, it is expected that those capabilities will be added into the language in a later release to enable vendor-lock-free introspection of safety projects.

Equivalent Tables

Equivalent tables show required and user-defined attributes of objects defined previously. This example may deviate from textual definitions and serves only for illustrative purposes.

Table 13. FMEDA

Project	Failure Rate (FIT)	Name	Element	Potential Faults	Potential Effect(s) of Failure	ISO 26262 Equivalent Fault/Error/Failure	Systematic or Random Failure?	Perm or Tran	Safety Related	PVSG	Potential Cause(s)	Current PSM	Current LSM	KFMC, RF
IP_A	4,447E+1	F001	FLAGS	Flag is faulty	Loss of data	Processing units: Registers::Stack overflow/underflow	Random	Permanent	true	true	TDDB	SM001	LSM001	95,45%
		F003	WP	WR logic is faulty	Loss of data	Processing units: Registers::Stack overflow/underflow	Random	Permanent	true	true	TDDB	SM002	LSM001	94,44%
		F005	RP	RP logic is faulty	Loss of data	Processing units: Registers::Stack overflow/underflow	Random	Permanent	true	true	TDDB	SM003	LSM001	94,44%
		F007	SRAM	Failure in SRAM bits	Incorrect data	Volatile memory::d.c. faults model (addr,data,control)	Random	Permanent	true	true	TDDB	SM004	LSM001	40,14%

1091 Table 14. List of SMs

Project	Name	Status	Safety Mechanism	Diagnostic or Avoidance?	Category	Error Response	Equivalent ISO 26262 Diagnostic	ISO 26262 DC	Default SM Type	Permanent KRF	Transient KRF	Permanent KMPF
IP_A	SM001	Active	Flag Logic Dup	Diagnostic	HW	HW Error Flag	Processing units: Registers::HW redundancy (e.g., dual core lockstep, asymmetric redundancy, coded processing)	High	Dup & cmp	95,00%	90,00%	0,00%
	SM002	Active	WR Logic Dup	Diagnostic	HW	HW Error Flag	Processing units: Registers::HW redundancy (e.g., dual core lockstep, asymmetric redundancy, coded processing)	High	Dup & cmp	95,00%	90,00%	0,00%
	SM003	Active	RD Logic Dup	Diagnostic	HW	HW Error Flag	Processing units: Registers::HW redundancy (e.g., dual core lockstep, asymmetric redundancy, coded processing)	High	Dup & cmp	95,00%	90,00%	0,00%
	SM004	Active	ECC	Diagnostic	HW	HW Error Flag	Volatile memory::Memory monitoring using	High	ECC	99,00%	99,00%	0,00%

							error-detection-correction codes (EDC)					
	LSM001	Active	LBIST	Diagnostic	HW	Abort	Processing units: Registers::Self-test supported by hardware (one-channel)	Medium	LBIST			95,00%

Table 15. List of FRs

Project	SPFM	LFM	PMHF	Type
IP_A	91,96%	92,74%	4,970E+0	Perm
IP_A	97,95%	N/A	1,786E-6	Tran

XII. Bibliography

[1] Accellera Functional Safety Working Group: White Paper, 2021 *https://accellera.org/images/downloads/standards/functional-safety/Functional_Safety_White_Paper_051020.pdf*

[2] ISO 26262:2018 Road Vehicles – Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems

[3] IEC 61508:2010 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems

[4] IEEE 1666-2011 – IEEE Standard for Standard SystemC Language Reference Manual

[5] 1364-2005 – IEEE Standard for Verilog Hardware Description Language

[6] IEEE P2851 *https://sagroups.ieee.org/2851/*

[7] P1800 - Standard for SystemVerilog – Unified Hardware Design, Specification, and Verification Language

[8] IEEE 1685-2014 – IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows

[9] IEEE P1801 – Draft Standard for Design and Verification of Low Power, Energy Aware Electronic Systems (UPF)

[10] *https://sysml.org/*

[11] "The Data Model Resource Book," Len Silverston, Wiley

[12] *https://www.gleek.io/blog/conceptual-data-model*

[13] The Entity-Relationship Model – Toward a Unified View of Data (ACM Transactions on Database Systems, Vol. 1, No. 1, 1976) - *https://dspace.mit.edu/bitstream/handle/1721.1/47432/entityrelationshx00chen.pdf*

[14] *https://www.guru99.com/data-modelling-conceptual-logical.html*

[15] “Hazard Analysis Techniques for System Safety,” Clifton A. Ericson, Wiley