international Engineering Safety Management

Good Practice Handbook

Application Note 10

HDL in FPGA Applications Guidance for Using EN50128

Published on behalf of the International Railway Industry

by Technical Programme Delivery Ltd

Issue 01, December 2018

We are grateful to the organizations listed who have supported iESM in various ways:

Cover pictures © 2012 Paul Cheeseman

© Technical Programme Delivery Ltd 2018

Disclaimer

Technical Programme Delivery Limited (TPD) and the other organizations and individuals involved in preparing this handbook have taken trouble to make sure that the handbook is accurate and useful, but it is only a guide. We do not give any form of guarantee that following the guidance in this handbook will be enough to ensure safety. We will not be liable to pay compensation to anyone who uses this handbook.

In particular, the examples given in this document are unlikely to be the same as your situation. They are intended to share good practice but must be considered in their context.

Acknowledgements

This Application Note has been written with help from the people listed below.

- Mike Castles
- Paul Cheeseman
- Dr Roger Chen
- Dr Bruce Elliot
- Paul Fletcher
- Greg Newman
- Gab Parris
- Garth Topham

Some of these people worked for the organizations listed below whilst other contributed individually.

- Ricardo Rail
- Rio Tinto
- Technical Programme Delivery Group
- TCT
- WSP

This guidance does not necessarily represent the opinion of any of these people or organizations.

© Technical Programme Delivery Ltd 2018

Contents

© Technical Programme Delivery Ltd 2018

1 Introduction

This Application Note Annex is a component of the international Engineering Safety Management Good Practice Handbook, or 'iESM', for short. The handbook describes good practice in railway Engineering Safety Management (ESM). It covers both projects that build new railways and projects that change existing railways. The iESM handbook is structured in three layers:

- Layer 1: Principles and process
- Layer 2: Methods, tools and techniques
- Layer 3: Specialized guidance

Layer 1 contains Volume 1. Volume 1 describes some of the safety obligations on people involved in changing the railway or developing new railway products. It describes a generic ESM process designed to help discharge these obligations.

Volume 2 provides guidance on implementing the generic ESM process presented in Volume 1 on projects. Volume 2 belongs in the second layer. At the time of writing, Volume 2 was the only document in the second layer, but further volumes may be added to this layer later.

The third layer comprises a number of Application Notes providing guidance in specialized areas, guidance specific to geographical regions and case studies illustrating the practical application of the guidance in this handbook.

The structure of the handbook is illustrated in Figure 1.

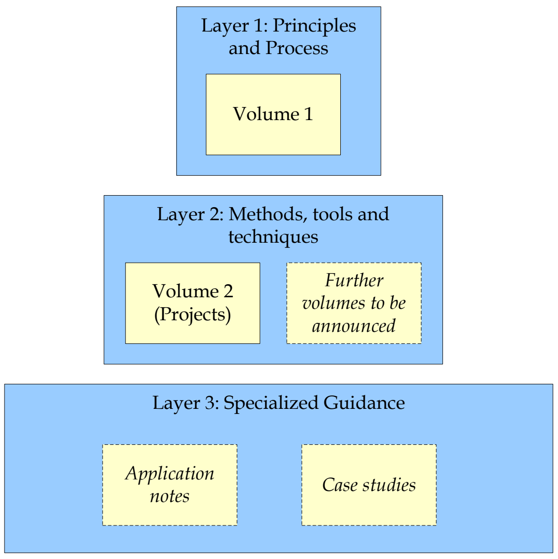

Layer 1: Principles and Process
Volume 1
Layer 2: Methods, tools and techniques
Volume 2 (Projects)
Further volumes to be announced
Layer 3: Specialized Guidance
Application notes
Case studies

Figure 1 The Structure of iESM Guidance

If you have any comments on this Application Note or suggestions for improving it, we should be glad to hear from you. You will find our contact details on our web site, www.intesm.org. This web site contains the most up-to-date version of this Application Note. We intend to revise the handbook periodically and your comments and suggestions will help us to make the Application Note more useful.

© Technical Programme Delivery Ltd 2018

2 Software Development in Engineering Safety Management (ESM)

2.1 Introduction

The backdrop to this Application Note (AN) is in terms of the relevance and application of the EN50128:2011 [1] standard to Hardware Description Language (HDL) software development as part of a railway control or protection system.

Specifically, the note sets out to review the EN50128 [1] design activities and general principles and contrast these against the specification and design activities of HDL generated code for gate array applications.

2.2 Scope

At present the scope of this AN is limited to a general introduction of HDL and uses comparisons with tried and trusted established software development principles for conventional microcontrollers as captured in the EN50128 [1].

The scope hence considers whether guidance in that standard can be applied to the generation of HDL code at a high-level, and if not, what considerations need to be made to achieve a similar level of robustness in the case where EN50128 [1] requirements are deemed not applicable.

This AN is intended for a manager or engineer that has been assigned responsibility for ensuring software development as part of railway control and protection systems and uses the elements of EN50128 [1].

This AN is not intended as detailed guidance for experienced FPGA design assessors or software specialists. It assumes a basic (but not advanced) appreciation of FPGA devices and programmable logic and software development including engineering processes and methods.

It should be noted that this AN offers a comparison only between development processes as outlined in the EN50128 [1] and those most commonly used in FPGA system development. It does not set out to provide a detailed level of instruction in relation to the FPGA lifecycle, as that would constitute a much broader industry-wide and detailed review, and which is presently on-going.

This document should be read in conjunction with the other Application Notes of the iESM Guidance and the additional reading material identified in Section 3.3.3.

© Technical Programme Delivery Ltd 2018

3 EN50128:2011

3.1 General

Software in general terms defines a collection or sequence of instructions, rules and data that will instruct, or be implemented on, (computer) hardware. The software in this generality is often contained by the notion of a 'program'.

EN50128:2011 [1] focuses on methods and tools used to develop software to meet the demands for safety integrity but the standard is not too specific on what constitutes 'software'. In essence, EN50128 [1] concerns safety-related 'software' being developed for conventional microcontrollers (programmable electronic systems) and deployed as part of wider railway control systems.

However, owing to limitations experienced in conventional developments, and in the modern microelectronic world, new silicon-based technologies (such as for ASICs and FPGA) have emerged offering much higher versatility and application flexibility, and employing new and different techniques in terms of their coding and testing (verification). And although EN50129 recognises that development and validation processes should *attempt to follow the EN50128 and its Annexes as far as practically possible,* full compatibility (mainly related to lifecycle phase sequencing) is generally not possible due to the fact that some operations are performed in a different order comparatively to the normal cycle as defined in EN50128 [1].

Consequently, the rapid rise of the new technologies and their growing potential for application in the railway environment creates a 'gap' that is still to be bridged, and for which EN50128 [1] is still providing guidance but is now judged as being either limited or highly restrictive in terms of the practicalities of performing various tasks which may no longer apply. Clearly, the scope for addressing the impact of new technology and the software elements they employ requires a far more technical, in-depth and industry-wide debate on the said technologies and in terms of how their design techniques differ and what constitutes good development practice.

The objective of this AN is therefore to commence setting out the fundamentals in relation to HDL and its use in FPGA developments and to help identify supporting guidance that can be brought to bear in such instances.

EN50128 [1] classifies software in its section 3.1.31 as:

Software *- intellectual creation comprising the programs, procedures, rules, data and any associated documentation pertaining to the operation of a system.*

However, there are other software (programmable logic) developments which will drive functional behaviour of a system similar to that of conventional software and which raises the question should EN50128 [1] broadly apply, and if not, then what

© Technical Programme Delivery Ltd 2018

assessment is expected in relation to programmable logic devices being deployed to control safety critical and safety related hardware in the railway environment?

Examples of this software development can be found in the modern and widely applied Field Programmable Gate Array (FPGA), Application-Specific Integrated Circuit (ASIC), Application-Specific Standard Part (ASSP) and Complex Programable Logic Device (CPLD) technologies which have software (or programming) development elements within their design lifecycle.

For the purposes of this AN, the FPGA is taken as the 'example' technology to asses in relation to the general assertions of the EN50128 [1] standard, but the observations used here can be applied to any similar programmable device that may form part of a sub-system or system.

3.2 Field-Programmable Gate Array (FPGA)

3.2.1 Overview

A FPGA is a re-programmable/re-configurable logic device intended to be fully configured after its production. It offers a highly versatile, high speed, and cost-effective solution in modern hardware developments and particularly useful for developing complex applications. Its origins stem from the early Programmable Read-Only Memory (PROM) and Programmable Logic Device (PLD) revolution of the eighties.

Fundamentally, FPGA technology can be thought of as the software development for hardware design. It is a modern integrated circuit designed to be 'programmed' for application by an end-user and can be configured (and re-configured) to implement a wide spectrum of digital functions and is deemed relatively straightforward to de-bug. For the purposes of this AN, the end-user will take the form of a designer required to implement a set of requirements including instructions to hardware devices that are specific to, for example, a control or safety protection system that may be deployed into the railway environment.

To make a comparison between software design in FPGA technology and conventional microcontroller development, it is necessary to deconstruct the tasks associated with FPGA programming, that uses the Hardware Description Language (HDL) and automated tools, as opposed to programme development language and conventional tools for software in microcontrollers, and see how these compare to the general requirements, principles and expectations as identified in the EN50128 [1].

The FPGA is essentially an array of programmable logic blocks and interconnects that can be 'wired' in such a way so as to provide a desired logical outcome based on a pre-determined set of inputs. As noted, they are also highly versatile in that they can be configured (and re-configured) to perform highly complex parallel functions

© Technical Programme Delivery Ltd 2018

or simple functions alike, and which gives them far reaching application options well beyond conventional microcontroller circuits. Consequently, demonstrating their safety properties and ensuring correct functional operation under all prevailing input conditions represents a considerable challenge.

Current applications see them being used and valued as part of defence, automotive, space, industrial, medical and communications projects world-wide. Fundamentally, and in regard to software engineering, the engineer is not writing code specifically that will be compiled into commands to be implemented sequentially on a microprocessor. Instead, the 'code' generated is describing hardware and its functionality. This code can then be synthesised and later fabricated direct onto FPGA technology.

3.2.2 FPGA Architecture

To understand 'software' as applied to FPGA technology, it is first important to understand standard FPGA architecture and the configured components that will implement logic. The following is hence a simplified introduction to the elements of the FPGA.

The FPGA will contain, to some degree, programmable logic blocks and reconfigurable interconnections, a bank of high-speed input/output (I/O) pins, and (in some cases where high demands are placed on memory) configurable Static Random Access Memory (SRAM) devices. A basic array architecture is indicated below.

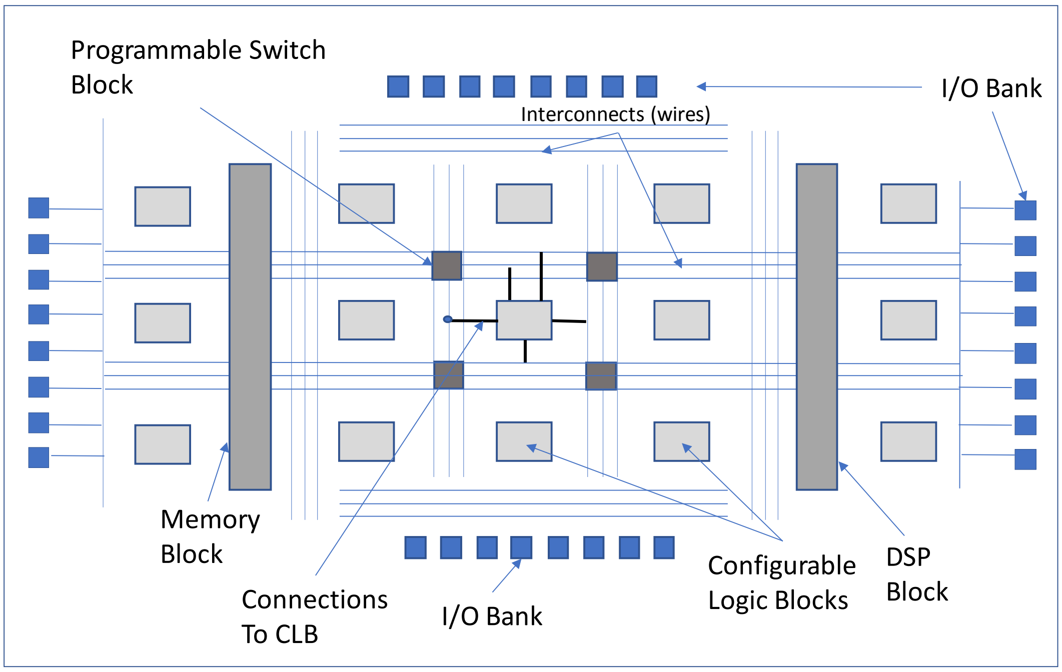


Figure 2 Array-Based Simplified FPGA Architecture

© Technical Programme Delivery Ltd 2018

The key physical elements and functions of a simple array are (i.e. may contain some or all):

- Configurable Logic Blocks (CLB) – which will implement the required logical functions (e.g. via Look-Up Tables (LUT), logic devices - flip-flops, registers, etc (various), MUX devices, see also figure 3).
- Interconnections – routing channels that provide flow direction and will connect logic functional elements, including clock routing network.
- Programmable Switches – which connect a logical element to an interconnect wire or one interconnect wire to another.
- I/O bank – user selectable, groups of pins that communicate the array input/output externally to/from other devices on a PCB, including dedicated input clock pins.
- Memory Blocks – which provide fixed embedded memory block functions within the array (for larger memory structures).
- DSP Blocks – embedded multipliers and adders, e.g. which can be incorporated for aspects of signal processing, and which will be application dependent etc.

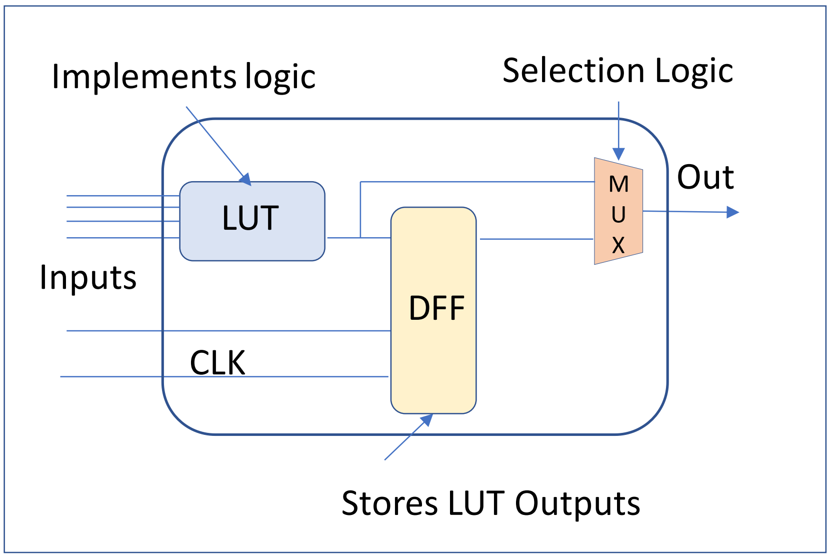


Figure 3 Elements of a Simple Logic Block

3.2.3 Hardware Description Languages (HDL) and FPGA

For FPGA circuits, the Hardware Description Language (HDL) is used to describe the logical aspects of the circuits and their behaviour. HDL is essentially a software programming language used to model hardware but unlike conventional (procedural) software, it will contain an explicit notion of time.

© Technical Programme Delivery Ltd 2018

The two principal HDLs are VHDL and Verilog and both will implement Register Transfer Level (RTL) abstractions at various levels. For this discussion, VHDL is considered because of its slightly wider use and application appeal across all industries. Verilog code structure is also based on C, while VHDL is based on Ada.

The differences between the two most prominent HDLs is in terms of dataflow, and regarding behavioural and structural dissimilarities with, for example, VHDL being strongly typed, more verbose and self-documenting with complex datatype and library management capabilities unlike Verilog, and which is also case sensitive.

3.2.4 Standards

In order to prevent potentially unsafe attributes of HDL code from leading to unsafe design issues, HDL coding standards (such as for the aviation industry DO-254) have been developed. RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware provides levels of guidance for the development of airborne electronic hardware, but the principles can be generally adapted for other industries.

The DO-254 standard provides compliance expectations for the design of complex electronic hardware in airborne systems and importantly this includes devices like Field Programmable Gate Arrays (FPGAs), Programmable Logic Devices (PLDs), and Application Specific Integrated Circuits (ASICs) and is the industry counterpart to software standard DO-178B/EUROCAE ED-12B.

The DO standard also notes five levels of compliance, A through E, which depend on the effect a failure of the hardware, with Level A the most stringent, defined as "catastrophic" while a failure of Level E hardware will not affect the system safety and the core principles can be used for other industry applications where system safety is deemed paramount.

Relevant standards when using VHDL include:

- IEEE1076 (and its sub-standards): which provides the formal notation for VHDL supporting development, verification, synthesis and testing of hardware design;
- IEEE1164: which provides for example standard types and functions library, (STD_LOGIC_1164) and packages;
- IEC61508: which provides general practical information for the techniques and measures for the design of ASICs from which some specific activities can be applied to FPGA technology;
- EN50129 [3]: which provides high-level and general guidance on how to apply and consider the application of EN50128 [1] to FPGA technologies;

© Technical Programme Delivery Ltd 2018

- EN50129 [3] (Annex F) – which provides application guidance concerning programmable components (PC), and in relation to complex/non-complex development processes, techniques, measures and relation to SIL;
- Def-Std-00-54: Requirements for Safety Related Electronic Hardware in Defence Equipment – (This standard has been superseded since Issue 3 of Def Stan 00-56 Safety Management Requirements for Defence Systems) which captures the requirements and guidance of the UK Ministry of Defence on the procurement, analysis, development and operation of safety-critical systems.

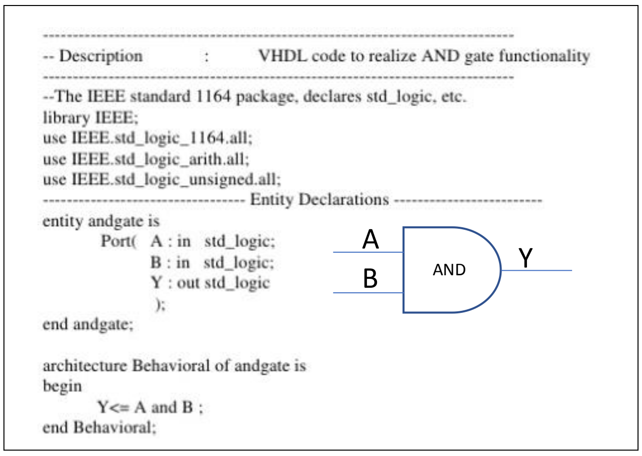
3.2.5 VHDL

Very high-speed integrated circuit HDL (or VHDL) is a dataflow language, as defined in IEEE Standard 1076, and allows for a system design to be modelled and verified (simulated) before synthesis and implementation tools formally translate the design into actual hardware. Two key advantages of VHDL are its reusability and its ability to describe concurrent (parallel) system operations.

In basic form the design items of VHDL consist of 'entities' which describe the symbols, generic (constants) and port (I/O) declarations, an 'architecture' which contains details of the logic and intended functions and key timings, a 'configuration' which is used to associate an architecture with an entity and a 'package' and which calls up re-usable information (e.g. libraries (i.e. pre-defined data and operator types (Boolean, time, character, string, real etc)).

VHDL also offers an extensive library of modules and can handle multiple architectures and configurations (using the same entities) as part of an integrated system and may also contain simulation constructs (i.e. testbench code) to assist in verification. A major advantage of VHDL over Verilog is that VHDL has a full 'type' system allowing circuit designers to write much more structured code.

A simple example of VHDL code expressing an 'AND' gate would be:



```
-------------------------------------------------------------------------------
-- Description          :       VHDL code to realize AND gate functionality
-------------------------------------------------------------------------------
--The IEEE standard 1164 package, declares std_logic, etc.
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
---------------------------------- Entity Declarations ------------------------
entity andgate is
        Port(   A : in   std_logic;
                B : in   std_logic;
                Y : out std_logic
                );
end andgate;

architecture Behavioral of andgate is
begin
        Y<= A and B ;
end Behavioral;
```

Figure 4 Basic HDL Code for 'AND' logic

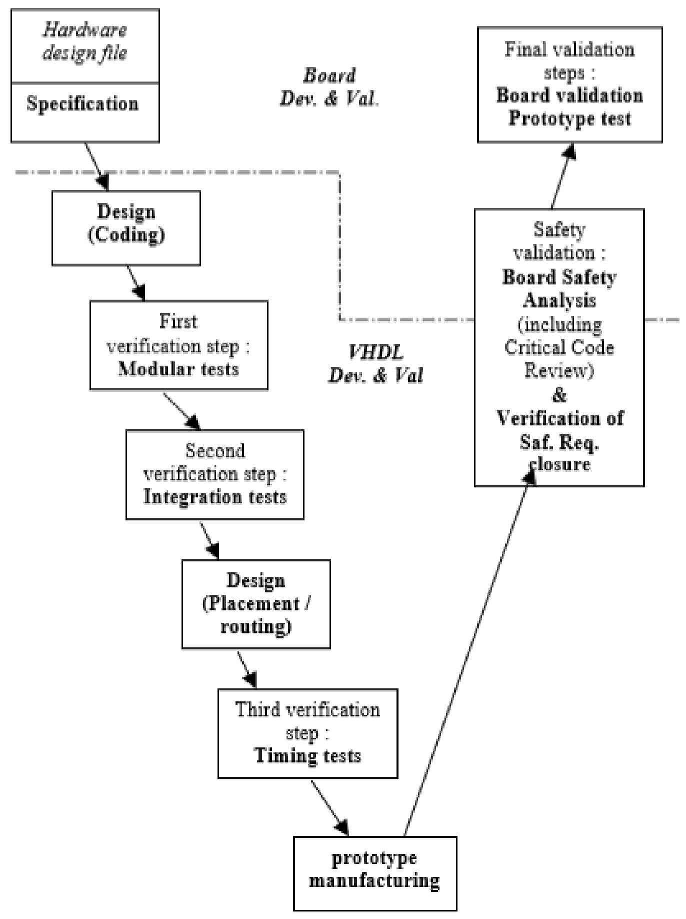
© Technical Programme Delivery Ltd 2018

3.2.6 FPGA Design Flow

The general design flow commences with the capture of the hardware functional requirements into a specification (and determination of any specific safety functions thereof). Figure 5 indicated below is the 'example 11' (taken as an extract from the withdrawn Railway Applications — Communication, signalling and processing systems — Application guide for EN50129, Part 2: Safety assurance, PD CLC/TR 50506-2:2009 [2]).

PD CLC/TR 50506-2:2009

– 25 – CLC/TR 50506-2:2009



Hardware design file
Specification
Board Dev. & Val.
Final validation steps : Board validation Prototype test
Design (Coding)
First verification step : Modular tests
VHDL Dev. & Val
Safety validation : Board Safety Analysis (including Critical Code Review) & Verification of Saf. Req. closure
Second verification step : Integration tests
Design (Placement / routing)
Third verification step : Timing tests
prototype manufacturing

Figure 11 – Example of development process for VLSI components (FPGA, EPLD, etc.)

Figure 5 FPGA Development Flow (typical)

Figure 5 identifies a relatively sequential development activity that translates the requirements and developed code into hardware (board level) for final verification. This general flow can be compared against the expected conventional design flow, for example, as identified in EN50128 [1], (Figure 4, 5.3.2.14) and extracted below as Figure 6.

© Technical Programme Delivery Ltd 2018

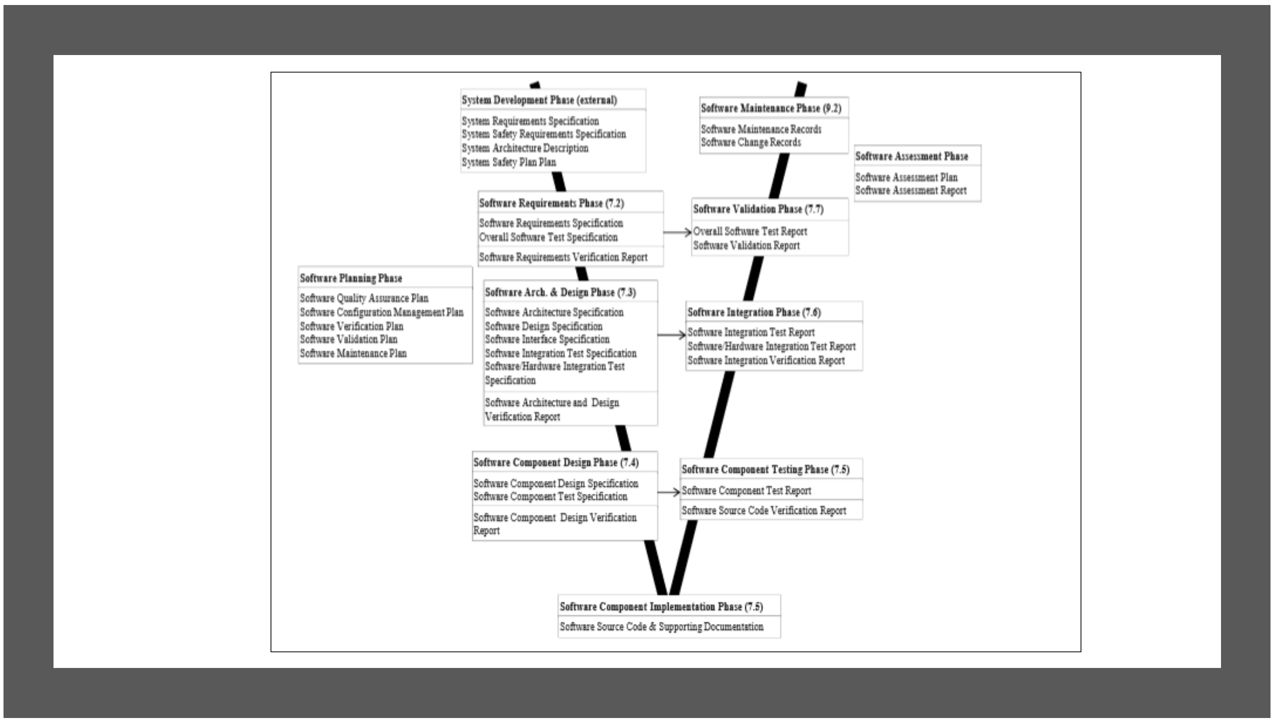

System Development Phase (external)
System Requirements Specification
System Safety Requirements Specification
System Architecture Description
System Safety Plan Plan
Software Maintenance Phase (9.2)
Software Maintenance Records
Software Change Records
Software Assessment Phase
Software Assessment Plan
Software Assessment Report
Software Requirements Phase (7.2)
Software Requirements Specification
Overall Software Test Specification
Software Requirements Verification Report
Software Validation Phase (7.7)
Overall Software Test Report
Software Validation Report
Software Planning Phase
Software Quality Assurance Plan
Software Configuration Management Plan
Software Verification Plan
Software Validation Plan
Software Maintenance Plan
Software Arch. & Design Phase (7.3)
Software Architecture Specification
Software Design Specification
Software Interface Specification
Software Integration Test Specification
Software/Hardware Integration Test Specification
Software Architecture and Design Verification Report
Software Integration Phase (7.6)
Software Integration Test Report
Software/Hardware Integration Test Report
Software Integration Verification Report
Software Component Design Phase (7.4)
Software Component Design Specification
Software Component Test Specification
Software Component Design Verification Report
Software Component Testing Phase (7.5)
Software Component Test Report
Software Source Code Verification Report
Software Component Implementation Phase (7.5)
Software Source Code & Supporting Documentation

Figure 6 Extract of EN50128 Illustrative Software Development Lifecycle 2

However, consideration of EN50129 [3] also highlights the following key statements:

- *Development and validation processes should follow EN50128 and its Annexes as far as practically possible. Full compatibility (mainly related to phases sequencing) is generally not possible due to the fact that some operations are performed in different order compared with the normal cycle defined in EN50128.*
- (Comparing with EN 50128): *it appears that the low part of the V-cycle (SW architecture & Design Phase, Code Phase, SW Module Design Phase, SW Module Testing Phase, SW Integration Phase) has a good correspondence of VHDL development, whereas the upper part (SW Requirements Spec Phase, SW/HW Integration Phase, SW Validation Phase) is in correspondence with the design and validation activity at the board level.*
- *Phases and operations as defined in EN50128 should have equivalent activities and document reports in relation with the apportioned SIL both at board level and at VHDL level. This correspondence should be clearly traced in final board safety report.*
- *In relation with development phases it is also note that all VLSI component design documents and intermediate files will join the HW documents in a normal configuration management process in order to allow possible design evolution and future inspection.*

An important remark may be added concerning the maintenance phase. In case of SW rework regression tests are not to be only partially done but completely performed as there can be no complete traceability against the cell's location in the

© Technical Programme Delivery Ltd 2018

VLSI component. In fact, the design and validation process would better be entirely automated in order to allow easy design evolution.

EN50129 [3] Annex F has also recently provided additional informative text (expanding on the above) for Programmable Component (PC) Logic in terms of how to apply a tailored set of specific techniques and measures against systematic failure and for a desired SIL, and then how to integrate programmable components as part of the safety architecture that this AN is yet to consider. An example of high SIL requirements, with hardware functionality expressed in terms of non-complex programmable logic is shown the standard's Table F.1 [3] where the expectation would be to perform exhaustive black-box functional tests combined with a rigorous hazard identification and failure analysis of the PC. However, it remains to be determined whether this combination of techniques and measures is entirely sufficient for all systems with functions having attributes of high SIL and low complexity.

Hence, at this stage of the AN development, the AN only starts at looking at the aims of the EN50128 [1] design flow and in terms of the development principles by way of 'comparison' that could be adopted in relation to FPGA developments.

3.2.7 FPGA Design Lifecycle

A further simplified flow model for FPGA is shown below in Figure 7 and which will be used for the remaining purposes of this AN to discuss comparative features.

© Technical Programme Delivery Ltd 2018

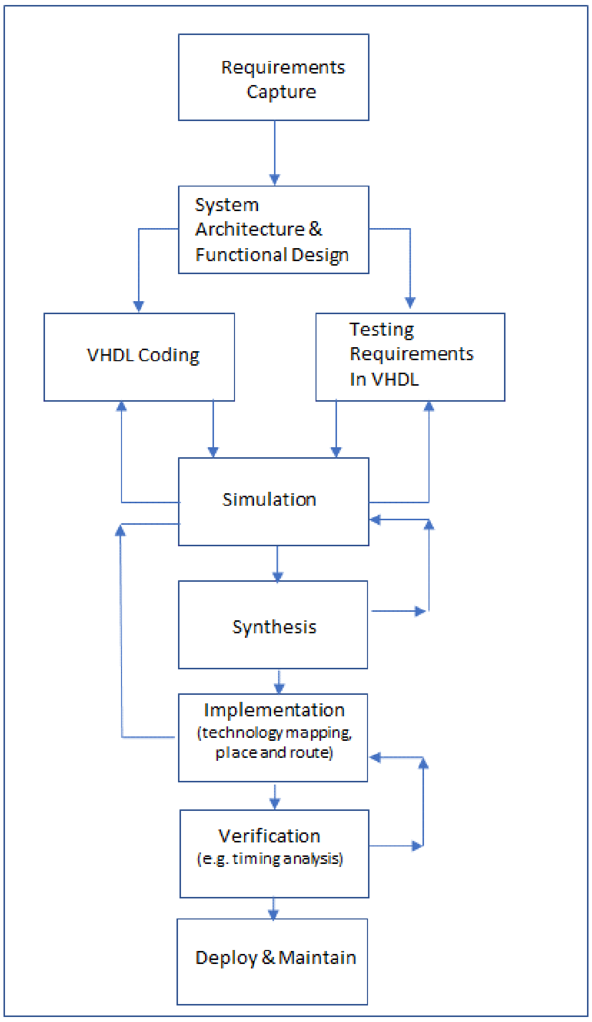

Requirements Capture
System Architecture & Functional Design
VHDL Coding
Testing Requirements In VHDL
Simulation
Synthesis
Implementation (technology mapping, place and route)
Verification (e.g. timing analysis)
Deploy & Maintain

Figure 7 Example of FPGA Development Flow Model

Using Figure 7, the FPGA design flow will follow a general approach that largely has the planned stages of:

- Requirements Capture:
  - Involves extraction (capture) of requirements and contained in a functional specification or a set of design description documents.
  - The range and scope of requirements documentation (i.e. what the design will do, and how it is intended to function) will depend on the level of functionality and complexity.
- System Architecture & Functional design:
  - Involves the analysis of the requirements including expected performance/behaviour, to determine the specific architecture, functions and interfaces needed.
  - Generation of test cases and definition of test environment to ensure the design will achieve requirements.
- HDL Design (VHDL Coding):

© Technical Programme Delivery Ltd 2018

- Generation of the device's specification requirements into a Hardware Description Language that can be compiled for simulation (using a proprietary simulation tool) and further manipulation.
    - (Note: when generating the specification requirements into a HDL, a coding standard check (similar to the one used for software implementation) is also often used).
- Simulation:
    - The HDL code, VHDL library information (and any required testbench code) are compiled to form the simulation file for modelling (and the stage being entirely independent of the target architecture).
    - *Verification stage* – Involves assessing the developed HDL for functional accuracy/correctness against, for example, behavioural models, state diagrams, truth tables, etc indicating what functionality is expected of the circuit (will simulate the logic functionality only as timing is not achievable at this level of abstraction).
    - With functional verification completed, the output will be HDL code file to be used for synthesis.
- Synthesis:
    - The HDL code is compiled and this stage will involve the translation and optimisation by a tool (the synthesizer) of the textural HDL output into the gate implementation or RTL netlist (note: synthesis is a process where functionality is being described schematically at the logic gate and board connection level)
    - Post-synthesis simulation can also be performed via a 'VHDL' output file that represents the synthesised output to feedback into the simulator for additional error checking (e.g. for a check for correctness of synthesis, etc)
    - *Verification stage* – this stage may include equivalency checking (formal verification) – i.e. that any design variants are functionally consistent between versions etc, board 'floorplan' design is consistent with e.g. FPGA technology constraints
    - The HDL code, VHDL library information and the intended FPGA technology library are then compiled to form the 'netlist' for the target device architecture.
- Implementation:
    - Involves the mapping of the post synthesis netlist onto the selected target FPGA hardware, i.e. to enable the performing of the 'place and route' (or the 'fitting') and which determines what specific FPGA selection resources that are available will be used and will be configured in relation to the digital circuit design (this results in the post-fitting netlist).

© Technical Programme Delivery Ltd 2018

    - Fitting in this regard will also have considerations in relation to device area usage, speed performance, power utilisation etc.
    - *Verification stage* – Post-fit simulation tools can assess and detect for errors.
    - The final configuration (the post-fitting netlist) being written to a file (*bitfile*) by a special programme bitstream generator for downloading and verification testing
- Verification (Timing)
    - The post-fit netlist can be used to assess if the design meets its timing specification requirements and other essential requirements.
    - *Verification stage* – E.g. Static timing analysis, post place and route, to test for violations relative to the clock or if there are specific timing considerations/constraints being imposed in relation to the technology or in terms of the specific application.
    - Further errors at this time are less probable, but it should be noted that proprietary FPGA editors (depending on the technology) also exist that will permit small scale change activity even post 'fitting' so as to correct inherent deficiencies, hence avoiding the need to re-implement. (note: this level of downstream manipulation will be entirely new to classical development processes of safety-related and safety-critical software).
    - (Note: not all devices can be subject to post-fit simulation particularly high density, highly complex devises and so other measures such as board level testing are often applied in relation to verification).
- Verification (post download)
    - Some final verification activity will need to be accomplished in circuit, but the technology offers faster resolutions if errors are exposed.
    - Generally, this part of the FPGA design process will mainly utilise chip de-bugging tools for the purposes of final validation.
- Deployment & Maintenance
    - Post verification, the design (bitfile) will be fabricated into the actual hardware, with final verification and then deployed and maintained as part of its lifecycle as with conventional software-based systems.

It should also be noted that the number of verification stages of the FPGA, and by the nature of FPGA devices and their manufacture, still remains quite high in comparison to other development lifecycle such as ASICs, but there are also many specialised proprietary tools available to assist in the design process for this technology that will permit analysing aspects related to pin allocation, performance, constraints, power management, utilisation, etc depending on the level of abstraction being undertaken which were not previously available as part of microcontroller design and applications.

© Technical Programme Delivery Ltd 2018

3.3 Comparison of an FPGA Development with EN50128

3.3.1 Similarities

As noted, this AN will compare the generic design flow between these types of software development and contrasts high-level techniques and measures between the identified stages only. However, it does not, at this stage, review specific programming techniques at the detailed level or any combinations of techniques deemed advisable in relation to SIL. This will require a much greater level of assessment.

The comparison is set specifically at a high-level from the requirements extraction through to integration of software with hardware and deployment into the field. A much more detailed analysis would be required to contrast the specific techniques, analysis and measures as indicated in the Annex A (Criteria for the Selection of Techniques and Measures) EN50128 [1].

The notion to be kept in mind in this flow example, is that software errors are essentially systematic. Hence, the robustness of the overall process is considered primary in regard to developing high integrity software. In this instance, FPGA makes much use of highly automated verification steps between stages in determination of potential errors and more so than conventional developments.

The microprocessor software-based development will have the conventional lifecycle stages as depicted in the EN50128 [1]: see also Figure 6. Each stage of the conventional lifecycle also has a set of specific inputs, outputs and requirements of that particular stage (in order to advance to the next stage) and which will generally include a stage verification analysis activity. The standard also identifies within the stage the level of rigor (in terms of Highly Recommended/Recommended activities) expected in relation to a desired level of integrity, and which will increase significantly between SIL0 and SIL4 developments. This aspect of the FPGA development is not explored at this stage.

At the earliest stages of development and planning and in relation to programming the FPGA, the situation is essentially no different. For FPGA there are semi-formalised and formal steps to be taken when translating the desired functional behaviour of the FPGA that could involve the use of a first stage pseudo code or hardware language (e.g. VHDL, Verilog), tools and other services that are used as part of the configuration.

Irrespective of the type of FPGA (i.e. the selected FPGA technology via a vendor), the end-user will first need to define the intended operation and functions of the hardware being controlled by the FPGA (the custom design), this could be in the first instant in the form of a visual (graphical) schematic but will then feature the use of an established HDL (e.g. VHDL) to describe the system. From there, other

© Technical Programme Delivery Ltd 2018

proprietary tools can be used to translate the output into e.g. Register Transfer Level (RTL) netlists for further processing.

Each stage of the conventional software development is now considered in Table 1 in relation to the FPGA development to see if there are any common features that could be identified and therefore would necessitate a similar or equivalent approach in FPGA development.

EN50128 Stage	FPGA Stage Equivalence (with EN50129 Equivalence[1])	Stage Similarities
Software Requirements	Requirements Capture (PC Requirements Phase, EN50129)	The principles to generate a consistent set of behavioural and other requirements will generally remain the same for both with a concise Software Requirements Specification (SRS) being the objective
Software Architecture Software Design Software Component Design	Architecture/Functional Design (PC Architecture and Design Phase, & Logic Component Design Phase, EN50129)	The principles can largely remain the same for both with a concise software (and component) architecture and design definition (e.g. state diagrams) being the objective
Implementation & Testing	HDL Design, Simulation, Synthesis, implementation (technology mapping) (PC Logic Component Coding Phase, EN50129)	Post design verification, Code will be developed and tested in both cases. However, aspects of the Implementation for FPGA will have elements of both 'implementation & testing' and 'software integration' as per the conventional software development
Software Integration/ Software Validation	Implementation/Timing Analysis (i.e. post layout implementation) verification/validation (Logic Component Testing Phase, Physical implementation Phase, PC integration Phase, PC Validation Phase, EN50129)	EN50128 addresses software/software and software/hardware integration and testing. FPGA will feature mapping of the netlist onto target hardware for final timing testing and in-circuit

[1] The prEN50129:2016 equivalence is set against figure F.4 of [3]

© Technical Programme Delivery Ltd 2018

EN50128 Stage	FPGA Stage Equivalence (with EN50129 Equivalence[1])	Stage Similarities
		verification. Principles of ‘integrate and test’ will be common to both and with reference to validating system functionality against the core specification documentation. (it should be noted that both timing test and functional test need to be conducted. just like the software requirement test performed in the hardware target)
Software Deployment	Deployment & Maintenance	The general principles (re: release notes, configuration identification (baseline), application conditions, manuals, maintenance planning, change records, etc), governing the deployment and maintenance will be similar, however the mechanics may differ per application and per technology
Software Maintenance		

Table 1 Comparison of Development Stages - EN50128 V's FPGA

In this high-level comparison only, it is important to note that a number of steps in the conventional software development stages can readily be adjusted to meet the specific FPGA needs. This is based on what functions the FPGA will be implementing and how these will be applied and to what level of integrity on, for example, a railway application. This activity is currently external to this AN. Each lifecycle stage definition is now addressed individually in the following sections.

3.3.2 Software Requirements – EN50128

The conventional approach, after generating the relevant planning material (e.g. Software Quality Plan (SQP)) is to systematically determine a set of complete and consistent requirements (from for example, a formal requirement analysis that determines desired functions, operating modes, external interfaces, etc) that the software will be required to implement and which may contain specific safety and hardware-related requirements.

The collated requirements set will be expressed in a Software Requirements Specification (SRS) (or equivalent) and will contain (as a sub-set) all the properties of the software to be developed. The software integrity requirement will have been determined and identified as one of those specific requirements. The conventional

© Technical Programme Delivery Ltd 2018

process also identifies the Software Test Specification (STS) from the SRS with identified test cases.

For the FPGA design process (re: Requirements Capture), there is no reason as to why a similar level of detailed SMART requirements extraction, manipulation and configuration control, including initial test case determination and requirements verification should not be undertaken to an equivalent level of rigor as defined in EN50128 [1].

3.3.3 Software Architecture/Design/Component Design Specification – EN50128

For conventional developments, the principle is to develop an architecture that will deliver the elements as defined in the SRS. This is in part achieved by first decomposing high-level requirements into a series of sub-specifications and checking for consistency between them, and then verifying between each main stage.

This would usually involve developing additional lower level specifications such as the Software Architecture Specification (SAS), Software Design Specification (SDS), Software Interface Specification (SIS), Software Integration Test Specification (SITS), Software/Hardware Integration Test Specification and require a stage verification report.

Depending on complexity of the system, there are elements within each of the above documents which remain applicable in FPGA development but not all. For conventional software the component layer will represent the pre-coding stage and will have a supporting Software Component Test Specification (SCTS). For FPGA, test cases are often developed at that level of the decomposition.

Hence, depending on the system under development, the FPGA equivalent stage (System Architecture & Functional Design) may not require the same rigorous decomposition but this must be evaluated on a case-by-case basis and in relation to the intended application and its safety requirements. The lower series of specifications will also be addressing any specific safety requirements in the SRS and will take account of any known constraints. These principles should be preserved also in FPGA developments but set out an appropriate level.

In the conventional development case, the final step requires decomposition down to specification(s) at a component level that will purposefully guide the code generation activity and will include a verification of the component design specification at that level. The principle of clear and unambiguous specifications for both conventional and HDL code should ultimately be preserved, but it must also be recognised that this pre-coding activity for HDL is considered to be more fluid owing to the nature of the 'simulation' which can be continually used to compile the emerging design and check that the development is meeting the requirements.

© Technical Programme Delivery Ltd 2018

For FPGA, the overall design specification decomposition as outlined in the EN50128 [1] is not explicitly identified to any particular level, but it is clear that the I/O as noted in the EN can be taken into consideration for the FPGA design depending on what integrity requirements are demanded of it. So, for example, it would be essential still to ensure the consistency between any level of developed specification material at this stage (e.g. characterising the architecture, interfaces, design, test aspects, test environment) and to generate a report confirming how this consistency has been (or will be) suitably verified.

Primarily, the principles of ensuring accuracy and consistency between lower specification levels is entirely reusable in FPGA terminology and will depend on what level the FPGA development decomposition is to take, but the activity may not require a more detailed specification development than may already exist in the SRS.

3.3.4 Implementation and Testing - EN50128 [1]

With requirements decomposed into specifications and sufficiently verified, the conventional approach is to start to generate code from the lowest (i.e. component) level descriptions provided by the earlier stages. This formal coding activity has a direct relationship with the generation of code in the 'HDL code' stage for FPGA. In that both perform the programming (or writing of the actual code) stage and the testing of the code at various points in that development.

For EN50128 [1], the stage testing will be, for example, to verify the Software Component Design Specification(s) have been met as well as an indication of the level of test achieved against the particulars in the specification. The stage also features a formal Source Code Verification Report as a requirement for proceeding into the integration stage. The process will then perform the same (test, test coverage, verification) at the next higher level, and so on.

For the FPGA equivalent (where the configuration is specified via the HDL design), and because of the nature of HDL in that the output from the coding can be compiled for errors and immediately simulated via a suitable proprietary automated tool to check for overall correct functionality, the designer will make extensive use of simulation and models and at different levels of abstraction making the entire test process, error correction activity quicker and far more dynamic. Similarly, the post synthesised file can also be modelled to check for errors in the actual synthesis process.

Although the measures and techniques applied to coding and testing will have many dissimilarities, there are also many similarities in principle with conventional software testing in reference to the existence of formal test specifications, test scripts, test coverage metrics, and test recording elements. For FPGA, once the simulation is completed, synthesis can occur. Synthesis is the process of converting the 'specification' as defined in HDL code into potentially optimised logic gate schematics. A VHDL type file can also be generated at this time to feed directly back

© Technical Programme Delivery Ltd 2018

into the simulation tool to check for the correctness of the synthesis. (A simple example of the dynamic environment is when the post-synthesis netlist can be assessed for the actual technology propagation delays being reported by the chip manufacturer with the results back annotated onto the netlist prior to place and route). A VHDL coding standard check can be also automatically be performed by tools.

Process elements that require use of identified standards, adherence to quality and configuration and change control methods of the developed source code should similarly be treated in principle as being applicable to both developments as they represent best practice. However, what is not so clear is the level of rigor of change impact analysis, change notification, configuration control and regression test is applied and formally recorded in relation to FPGA developments, as these are more often likely to be controlled by 'internal' standards and practices.

In general terms, and in relation to FPGA, a similar mind-set for controlling change activity and the impact of change should be preserved across the development. However, a further and more detailed study is required to determine how this is traced and documented and ultimately controlled when using a high-level of automation, abstraction and manipulation implied during the FPGA stages. The current understanding is that these activities are recorded in the various file transactions that will form part of the Design File.

3.3.5 Software Integration/Software Validation – EN50128

Post EN50128 [1] implementation, the conventional process is to ensure the developed software and the physical hardware integrate correctly according to the detailed specifications. This activity features the notion of the 'composite whole' being the build-up of a 'system' by proving of individual component software elements at each stage to achieve the system software and then to test to ensure that the interaction of the system software with the intended hardware functions correctly.

For this stage, EN50128 [1] requires any change activity to be recorded, impact analysed and re-verification applied where appropriate. The final validation is to provide compliance evidence against the SRS with particular focus being on the safety requirements of the application, and success of which culminates in the release note. This level of final test can be achieved either on the target hardware or via simulation systems or both.

The FPGA equivalent would be covered by post FPGA implementation stage, where any technology mapping (place and route (fitting)), technology constraints and the desired logical functionality of the array can be tested on the proposed technology (using the vendor's specification).

© Technical Programme Delivery Ltd 2018

In reality, errors or issues determined at the latter stages of FPGA design should be subject to similar quality control and re-verification measures identified for conventional software, but this will depend on the nature of the change required. However, in principle, the aspects of robust change control, impact analysis and re-test and verification to requirements is largely no different, as previously mentioned, and should be replicated as good practice.

For both conventional and FPGA, the output of the final software is contained on a device that can be loaded either onto a microcontroller or FPGA chip using a suitable interface. FPGA requirements testing can also be performed at the level of in-circuit test, which will combine the software, hardware and VHDL together to test the correctness of FPGA requirements.

3.3.6 Software Deployment & Maintenance – EN50128

The conventional approach considers if the software will perform as expected in its final environment. Deployment will ensure that essential information supporting software installation, rollback, and diagnostics is provided (in detail) to the end-users with the appropriate manuals. Methodology and process for future enhancements, corrections and adaptations which will require layers of pre-planning will often be captured in the Software Maintenance Plan (SMP). Here, the notion is that software is deployed and loaded onto a hardware platform and modifications are strictly controlled.

The FPGA equivalent is similar in that there is a completed programming file output that can be transferred/downloaded (via flash ROM or through direct connection) onto the physical FPGA device so that it is used to programme that device on power-up. The principles of making changes to its 'bitfile' should be treated essentially as per the conventional software case, requiring rigorous configuration control and the use of an auditable SMP and with documented and robust processes.

The issue at this level is that FPGA offers a relatively straightforward approach to change that would not be supported by EN50128 [1]. This is mainly due to how HDL has developed and the use of support tools to aid and prove design which has permitted a level of re-work (through editor tools) at late design stages, and which does not necessarily get re-annotated back to the primary design documentation. In general, the principles of EN50128 [1] governing change and change control should be applied fully across FPGA design developments to ensure that the design remains fully traceable throughout its lifecycle.

© Technical Programme Delivery Ltd 2018

4 Summary

At this stage, the AN only considers the 'principles' of a conventional software design flow (EN50128) as compared with a development that aims to utilise HDL and FPGA.

Information on plausible techniques/measures in relation to SIL is currently available in Tables F.3-F.8 in EN50129 [3] and although considered essential, these measures remain to be fully substantiated in terms of their sufficiency.

Further information will be forthcoming to look specifically at the measures, analysis techniques as currently applied in EN50128 [1] software development and notified in EN50129 [3] and how they would either directly apply, translate or be superseded in relation to FPGA developments.

Further ANs may address the use of HDL in the development of programmable technology and provide specific application guidance.

© Technical Programme Delivery Ltd 2018

5 Document References

The following external references are used in this document:

(1) EN50128:2011, Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems, 2011

(2) Railway applications — Communication, signalling and processing systems — Application guide for EN50129, Part 2: Safety assurance, PD CLC/TR 50506-2:2009) – now withdrawn

(3) EN50129:2018 - Railway applications - Communication, signalling and processing systems- Safety related electronic systems for signalling

© Technical Programme Delivery Ltd 2018

6 Glossary

This glossary defines the specialized terms and abbreviations used in this Application Note annex.

Engineering Safety Management (ESM)	The activities involved in making a system or product safe and showing that it is safe. Note: despite the name, ESM is not performed by engineers alone and is applicable to changes that involve more than just engineering.
Hazard	A condition that could lead to an accident. A potential source of harm. A hazard should be referred to a system or product definition.
Safety	Safety is defined as "freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property, or damage to the environment."
System	A set of elements which interact according to a design, where an element of a system can be another system, called a subsystem and may include hardware, software and human interaction.
System Lifecycle	A sequence of phases through which a system can be considered to pass. A product may also pass through some of these phases.

Abbreviations:

AN	Application Note
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Part
CLB	Configurable Logic Block
CLK	Clock
CPLD	Complex Programmable Logic Device
DEV	Development
DFF	D Type Flip Flop (clock-controlled memory device)
DSP	Digital Signal Processing

© Technical Programme Delivery Ltd 2018

FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HW	Hardware
I/O	Input/Output
LUT	Look Up Table
MUX	Multiplexer
PC	Programmable Component
PCB	Printed Circuit Board
PLD	Programmable Logic Device
PROM	Programmable Read Only Memory
RTL	Register Transfer Level
SIL	Safety Integrity Level (per 50128 [1])
SMART	Specific, Measurable, Attainable, Realistic, Time-bound
SQP	Software Quality Plan
SRAM	Static Random Access Memory
SW	Software
VAL	Validation
VER	Verification
VHDL	Very High-Speed Integrated Circuit HDL
VLSI	Very Large Scale Integrating

© Technical Programme Delivery Ltd 2018

international Engineering Safety Management

Good Practice Handbook
Application Note 10

Published on behalf of the International Railway Industry
by Technical Programme Delivery Ltd