DEPENDABLE DESIGN FLOW FOR PROTECTION SYSTEMS USING PROGRAMMABLE LOGIC DEVICES

M. Kwiatkowski, B. Todd, CERN, Geneva, Switzerland

The 13th International Conference on Accelerator and Large Experimental Physics Control Systems. 10-14th October 2011, Grenoble, France.

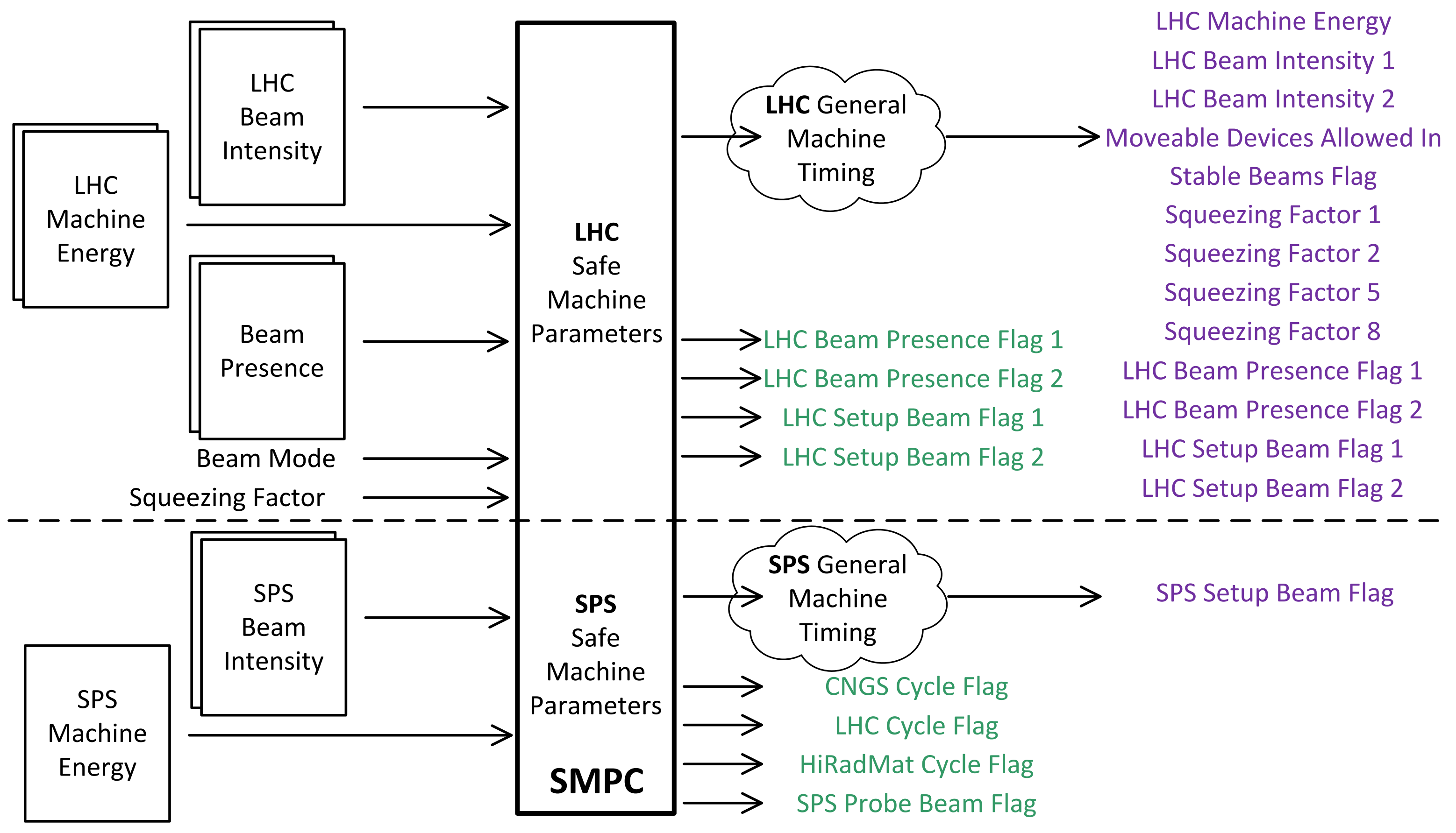
Abstract

Programmable Logic Devices (PLD) such as Field Programmable Gate Arrays (FPGA) are becoming more prevalent in protection and safety-related electronic systems. When employing such programmable logic devices, extra care and attention needs to be taken. The designer's confidence can be increased using techniques such as Formal Methods, exhaustive Hardware Description Language (HDL) code simulation and hardware testing. An example is given for one of the critical functions of the Safe Machine Parameters (SMP) system, used in the protection of the Large Hadron Collider (LHC) at CERN.

CERN is also working towards an adaptation of the IEC-61508 lifecycle designed for Machine Protection Systems (MPS), and the High Energy Physics environment, implementation of a protection function in FPGA code is only one small step of this lifecycle.

SAFE MACHINE PARAMETERS (SMP)

A complex Machine Protection System (MPS) has been designed to protect the LHC accelerator, a fundamental part of the MPS is the Safe Machine Parameters Controller (SMPC). For the correct protection of the LHC and its accelerator complex, several parts of the MPS require information about the machine's operational parameters. Values such as beam intensities, machine energies, squeezing factors, amongst several others, must be broadcast around the accelerator complex to correctly configure the MPS, and sent to the extraction interlock systems to ensure the correct interlocking of beam transfer between the Super-Proton Synchrotron (SPS) and the LHC.

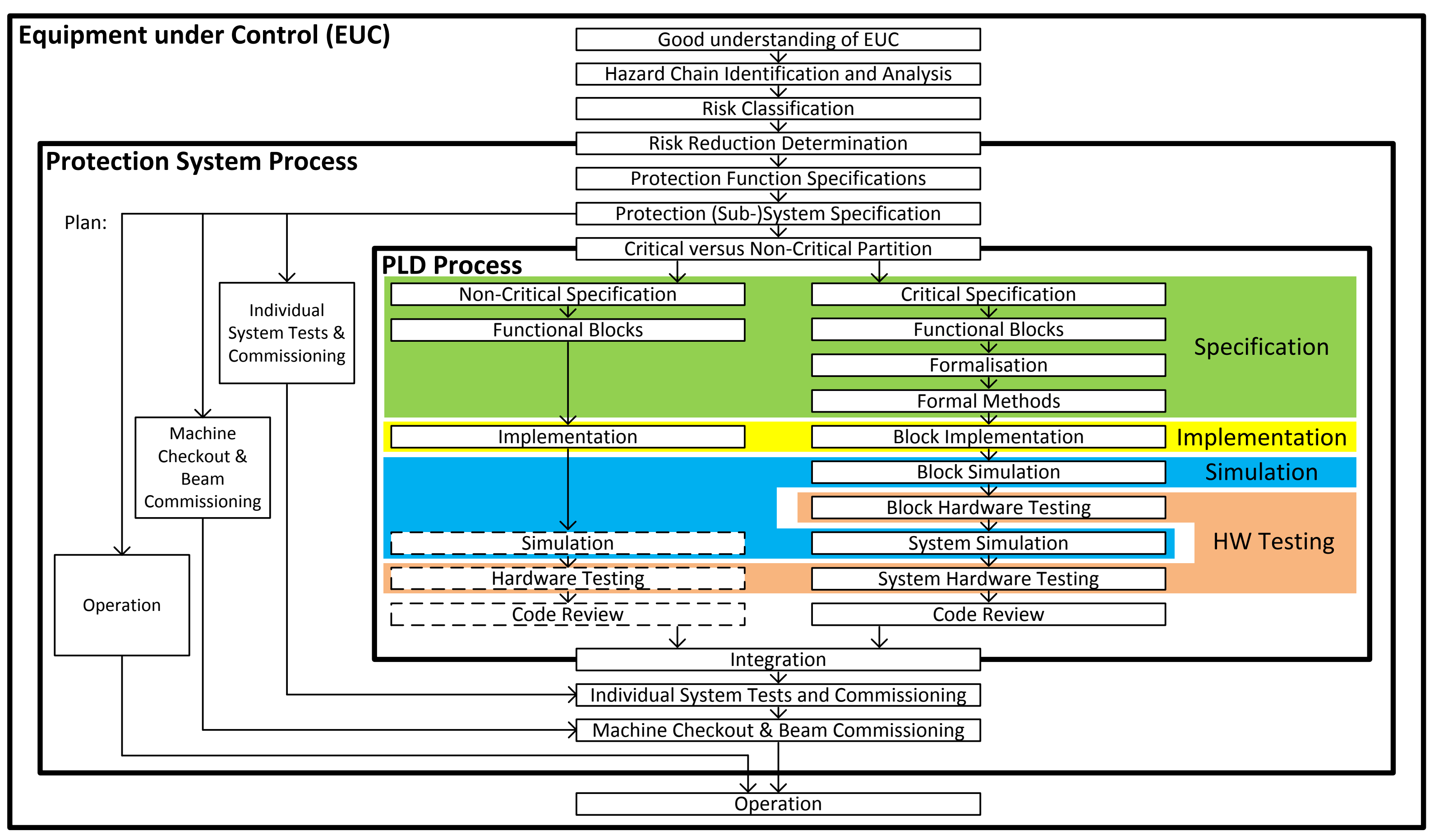


The outputs of the SMPC either go directly to the extraction interlock controllers, or are broadcast around the machines using the General Machine Timing (GMT) network. In realising this mission-critical function, the SMP system makes extensive use of PLDs.

MACHINE PROTECTION SYSTEM LIFECYCLE

During the development of the SMPC, and some of the other MPS subsystems, common themes and ideas have began to emerge. The combination of these ideas has led to the concept of a MPS Lifecycle (MPSL), based on the IEC 61508 overall system lifecycle. The MPSL concept allows a consistent approach to the evaluation of existing parts of the MPS, whilst at the same time providing a framework for the development of new protection systems.

The part of the MPSL which is relevant to the programmable logic device flow is that related to the implementation of so-called Protection Functions



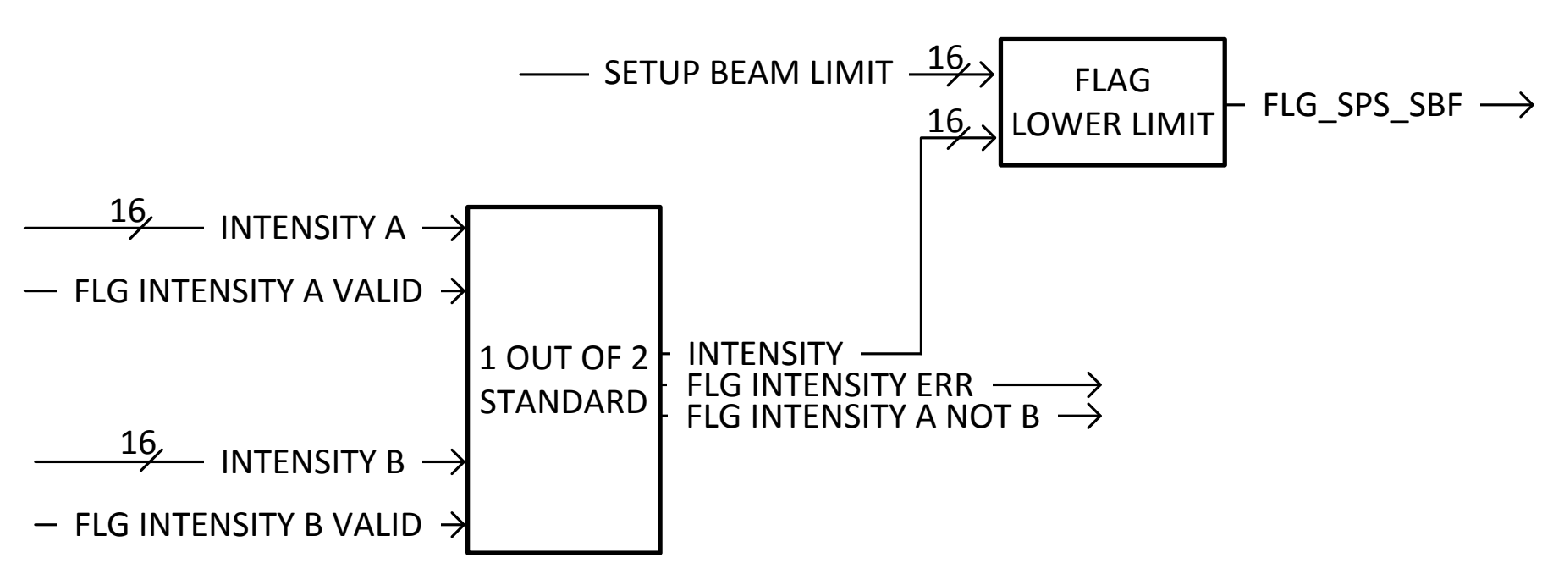
Equipment under Control (EUC)
Good understanding of EUC
Hazard Chain Identification and Analysis
Risk Classification
Risk Reduction Determination
Protection System Process
Protection Function Specifications
Protection (Sub-)System Specification
Critical versus Non-Critical Partition
Plan:
PLD Process
Non-Critical Specification
Functional Blocks
Critical Specification
Functional Blocks
Formalisation
Formal Methods
Specification
Implementation
Block Implementation
Implementation
Block Simulation
Simulation
Block Hardware Testing
Simulation
System Simulation
HW Testing
Hardware Testing
System Hardware Testing
Code Review
Code Review
Individual System Tests & Commissioning
Machine Checkout & Beam Commissioning
Operation
Integration
Individual System Tests and Commissioning
Machine Checkout & Beam Commissioning
Operation

PROGRAMMABLE LOGIC DEVICE PROCESS

The PLD design process starts with clear separation of what is part of the PFs from other non-critical functions. Functions that are part of the PF are to follow this PSL process

Specification phase

Each critical function must be decomposed into functional blocks, ones which lend themselves easily to analysis and understanding. In addition, each must be capable of being readily specified using a formal language. For example, the block diagram and its formal version of the SPS Setup Beam Flag (SBF) is shown on figure. The formal language used in this example is Predicate Logic. The key strength of this Predicate Logic formal language is in interpretation: if correctly written, it can only be understood by designers in a single way, which is not always the case for a traditional specification. The formal language also allows a formal verification of some design parameters, allowing a mathematical verification the completeness and the consistency of the specification.



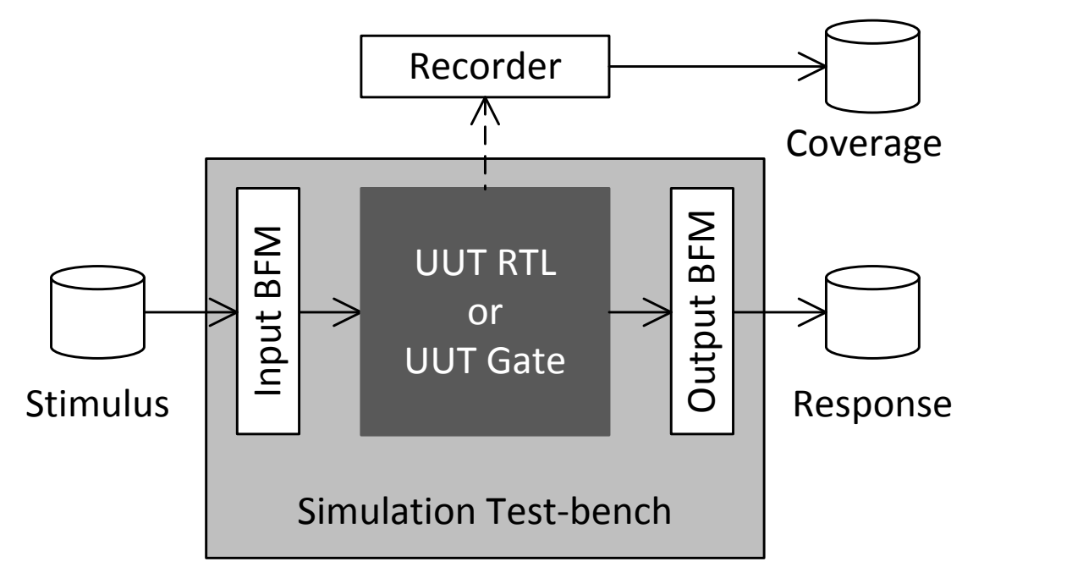
```
FAIL_SAFE_HIGH_INTENSITY : int := 6.5535 * 10^14;

SPS_SMPC_SBF (
   SETUP_BEAM_LIMIT : int,
   INTENSITY_A : int,
   FLG_INTENSITY_A_valid : bool,
   INTENSITY_B : int,
   FLG_INTENSITY_B_valid : bool,
   FLG_SPS_SBF : bool
   ) :=
      exists INTENSITY : int. (
      (INTENSITY =
         if FLG_INTENSITY_A_valid then INTENSITY_A;
         if else FLG_INTENSITY_B_valid then INTENSITY_B;
         else FAIL_SAFE_HIGH_INTENSITY;
      )
      AND
      FLG_SPS_SBF =
         if INTENSITY <= SETUP_BEAM_LIMIT then TRUE;
         else FALSE
   );
```

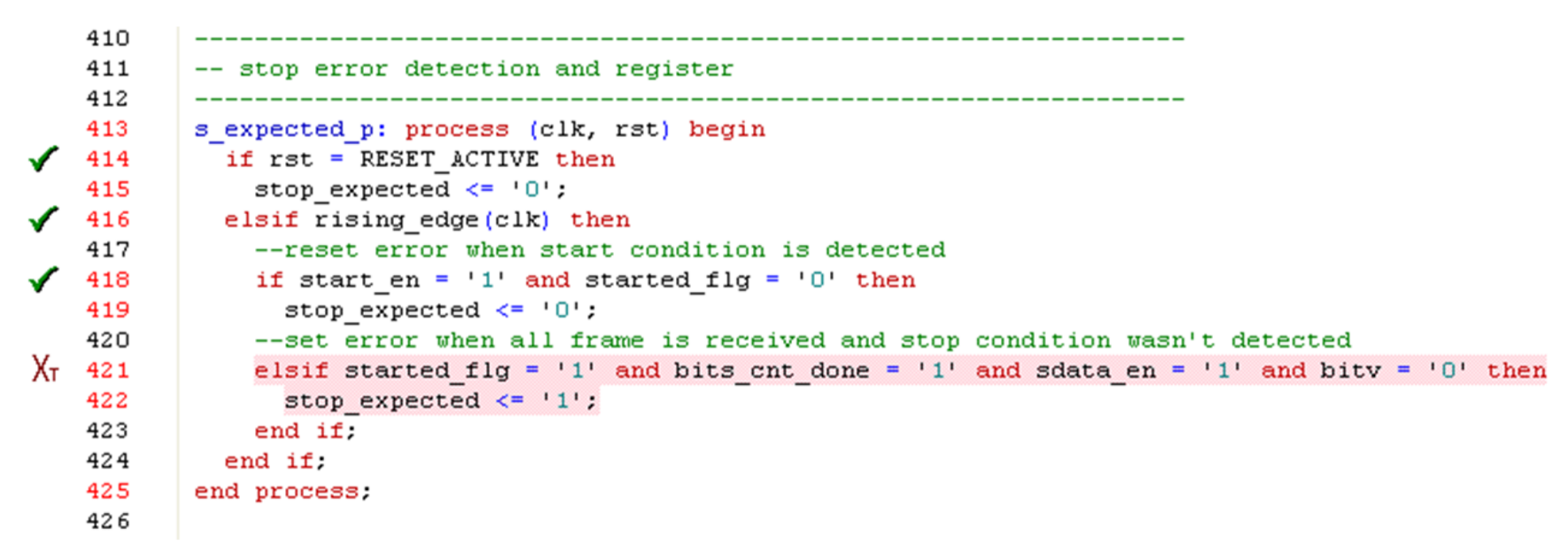
Implementation phase

Each of the smaller functional blocks which were conceived in the previous step are then implemented. The actual implementation phase is a very small part of the overall time spent on PLD firmware development. The vast majority of the time is spent on simulation and testing. Hardware Description Language (HDL) code is written to describe PLD function. The designer should always know what the expected result of the synthesis process is to be, this should be also verified using the output of synthesis tools. This kind of verification is much easier when the blocks are small and readily understood.

Software simulation phase

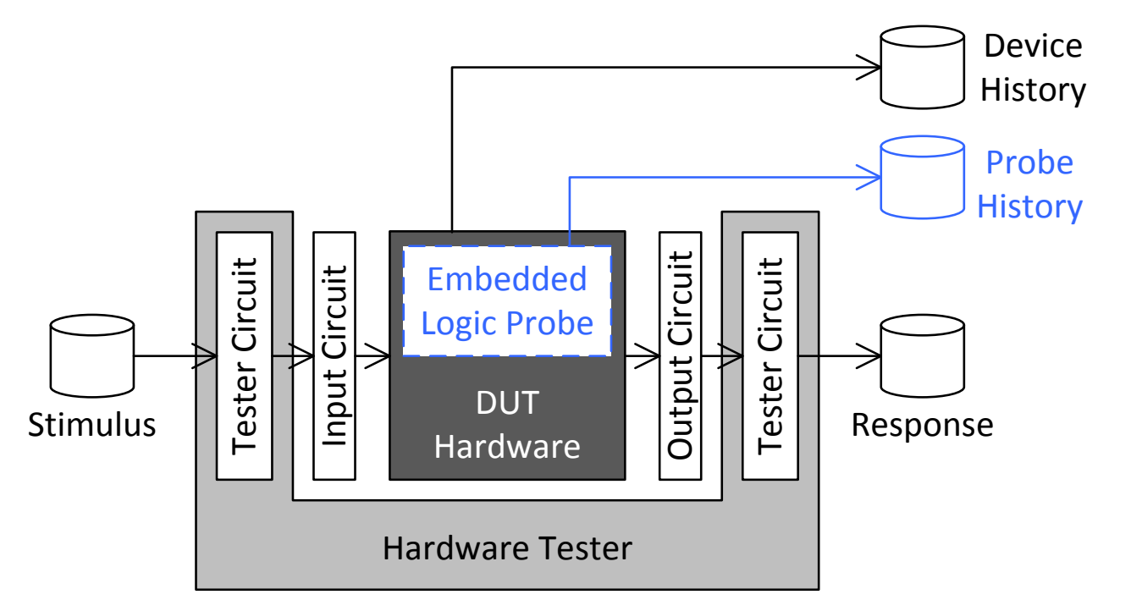
Software simulation is carried out on the block level as well as on the system level. Simulation with code coverage is a fundamental requirement for critical functions. A software test-bench is required, which should wrap the Unit Under Test (UUT) inside Bus Functional Models (BFMs), passing stimuli to the UUT and recording its responses. Behaviour which is not specified for the block or the sub-system can be detected and fixed, at the same time the test-bench should evolve to include new conditions as the weaknesses in code coverage are identified. It is preferred that a critical function achieves full code coverage. On the picture below the example of missing code coverage is presented.




```
------------------------------------------------------------------
-- stop error detection and register
------------------------------------------------------------------
s_expected_p: process (clk, rst) begin
  if rst = RESET_ACTIVE then
    stop_expected <= '0';
  elsif rising_edge(clk) then
    --reset error when start condition is detected
    if start_en = '1' and started_flg = '0' then
      stop_expected <= '0';
    --set error when all frame is received and stop condition wasn't detected
    elsif started_flg = '1' and bits_cnt_done = '1' and sdata_en = '1' and bitv = '0' then
      stop_expected <= '1';
    end if;
  end if;
end process;
```

Hardware testing phase

Once the blocks have been implemented and simulated, the real hardware can be generated. This is then tested using a dedicated hardware tester. Hardware testing is obligatory on the system level but optional at the block level. The hardware tester generates input stimulus and checks the response of the Device Under Test (DUT), in much the same way as the simulation test-bench, but this time using real signals. Its advantage over the software simulation is its speed and the possibility to introduce real distortions. A very useful hardware testing tool provided by device manufacturers is an Embedded Logic Probe. The probe is integrated with the design and finally programmed into PLD DUT. With this analyzer in place it is possible to record internal signals using a variety of trigger conditions. A typical setup with an embedded logic probe and the result of analysis is shown on figure below.



Device History
Probe History
Embedded Logic Probe
DUT Hardware
Stimulus
Tester Circuit
Input Circuit
Output Circuit
Tester Circuit
Response
Hardware Tester