DEPENDABLE DESIGN USING PROGRAMMABLE LOGIC DEVICES

A. Castañeda Serra*, M. Kwiatkowski†, B. Todd‡, CERN, Geneva, Switzerland

Abstract

Mission critical systems at the European Organisation for Nuclear Research (CERN) make extensive use of Programmable Logic Devices (PLDs) such as Field Programmable Gate Arrays (FPGAs) to implement their safety critical functions. The dependability of these safety critical functions is difficult to determine using traditional techniques. A robust approach is needed if PLD technology is to be accepted in mission critical systems.

This paper discusses techniques which are being developed and employed by CERN to give confidence in the use of PLDs in mission critical systems, the Safe Machine Parameter system development is used as an example.

SAFE MACHINE PARAMETERS (SMP)

For safe operation of the Large Hadron Collider (LHC), several systems require machine parameters that must be generated and distributed around the accelerator complex with high dependability (safety, availability and reliability). The Safe Machine Parameters (SMP) system is being developed to generate parameters from the machine and beam states, transmitting these parameters to the systems that require them.

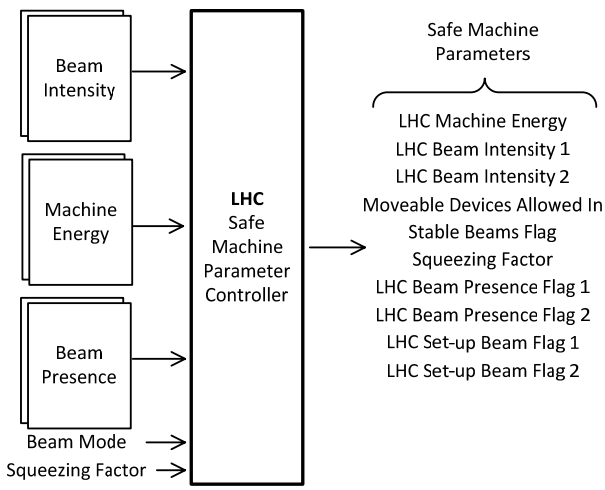

Figure 1: The LHC Safe Machine Parameter function.

In realising this mission-critical function, the SMP system makes extensive use of PLDs.

For 2009-10 operation of the LHC a *functionally correct* system has been realised.

For 2011 operation of LHC a *dependable system* is in development, with particular emphasis placed on the implementation of the PLDs.

*alejandro.castaneda.serra@cern.ch
†maciej.kwiatkowski@cern.ch
‡benjamin.todd@cern.ch

OPTIMISATION TECHNIQUES

The development and evolution of mission critical systems using PLDs, such as the SMP system requires four key areas to be thoroughly addressed during the system's design cycle [1]:

Separation of Critical from Non Critical

The design should be made in such a way as to minimise the implementation of the mission-critical path, anything which is not critical should be separated out, and preferably implemented in a completely separate device. The non-critical implementation should have no influence on the critical implementation, and the critical path should be as simple as possible, in a device that can easily meet the requirements with significant spare logic cells.

Monitoring the Critical Path

The design should allow the critical information and processes to be monitored, by providing signal taps and real time information. This allows the critical path integrity to be cross-checked during operation.

Simulation Test-Benches and Code Coverage

Realising the design should allow for the critical signal paths to be exercised using a purpose built test-bench, in addition, this test-bench should be written to be as flexible as possible, to increase the code coverage.

Stand Alone and On-Demand Testing

A specific hardware tester is required to ensure system performance in a laboratory environment. An automated tester verifies that design functionality is sound after modifications are made, and also verifies that the generated hardware matches the simulated function.

Once installed, the system critical paths should be testable on demand, in a safe way, in-situ. This does not necessarily require changes to the hardware implementation, but at least the effects of testing on the system operation should be considered.

SIMULATION AND CODE COVERAGE

Hardware Description Language (HDL) code, such as VHDL or Verilog, is written to describe the function of the PLDs. Simulation with code coverage is a fundamental requirement for mission-critical systems. Simulation with code coverage must be carried out at two levels of abstraction:

- Behavioural Simulation
- Gate-Level Simulation

Behavioural simulation does not always give results which match real hardware, on the other hand gate-level simulation matches the hardware, but is cumbersome and time consuming to carry out, as it uses the final design's

net-list and device specific primitives libraries. This makes gate-level simulation somewhat impractical especially during the design phase of a complex project.

In all cases a thorough test-bench is required, which should wrap the Unit Under Test (UUT) inside Bus Functional Models (BFMs), passing stimuli to the UUT and recording responses. The test-bench should evolve to include new conditions as the weaknesses in code-coverage are identified.

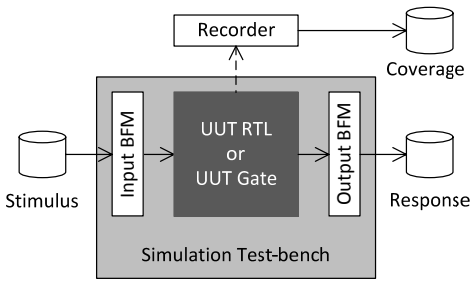

Figure 2: Simulation test-bench.

Code coverage is required in both cases, this encompasses numerous terms, the specific phraseology depends on the tools being used, but broadly speaking, code coverage falls into five categories [2]:

Statement Coverage

Statement coverage measures if and how often the signals and variables were assigned during the simulation. This is a basic measurement and the design should always have 100% statement coverage.

Branch Coverage

Branch coverage assures that all "if / case" branches have been executed. Empty branches can legitimately exist, so it is not always possible to cover branches to 100%. Exceptions such as this should be verified, and commented upon in the design code.

Condition Coverage

Condition coverage checks if all combinations of complex branch conditions were executed. For example "if A = B" would have all conditions covered when all four combinations of A and B have been checked, 00, 01, 10, 11. Implementation changes may be needed to permit the complete condition coverage.

Expression Coverage

Expression coverage checks the truth table of the expressions which are not used in decision constructs. It is possible to choose between two checking methods:

Exhaustive Coverage – Every expression combination is checked.

Focussed Expression Coverage – A sub-set of expression combinations are checked, giving a high degree of coverage, especially important if the expression has many inputs which would be impossible to exhaustively cover in a reasonable time.

Toggle Coverage

Toggle coverage determines if all bits of all signals have changed states, and is most appropriately used in *gate-level* simulations.

HARDWARE TESTING

Hardware testing is also required to validate the implementation, proving that final hardware conforms to specification. For this purpose a hardware tester should be realised.

Hardware Tester

The 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, logging real results.

The tester should be designed in such a way as to be able to invoke errors to verify the response of the hardware to these conditions.

Embedded Logic Probes

A useful hardware testing tool provided by device manufacturers is an Embedded Logic Probe (such as Xilinx's ChipScope or Altera's SignalTap). The probe is integrated with the design and finally programmed into PLD DUT. It uses device memory resources for recording selected internal or external signals, thus the critical device must have spare cells and memory, but with this analyser in place it is possible to record internal signals using a variety of trigger conditions. A typical setup with an embedded logic probe is shown below.

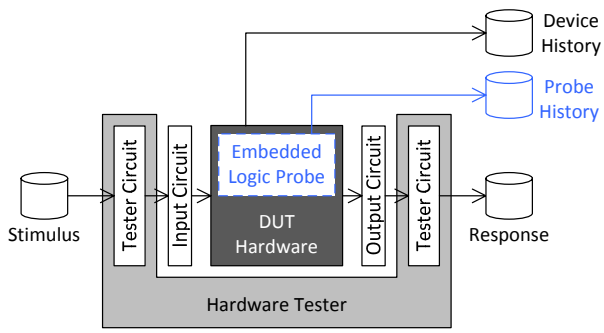

Figure 3: Hardware tester.

THE SMP RECEIVER

The LHC SMP system is being re-designed for 2011 operation, the first sub-system to be addressed is a receiver module (CISR). It is tasked with de-serialising raw intensity and energy data, checking the data integrity, and re-encoding the data. The old (2009-10) and new (2011) implementations are shown in Figure 4.

Separation of Critical from Non Critical

In both cases, two FPGAs are implemented, the first, a *control device,* is tasked with the critical function, and a second *monitor device* records and supervises the operation of the control device.

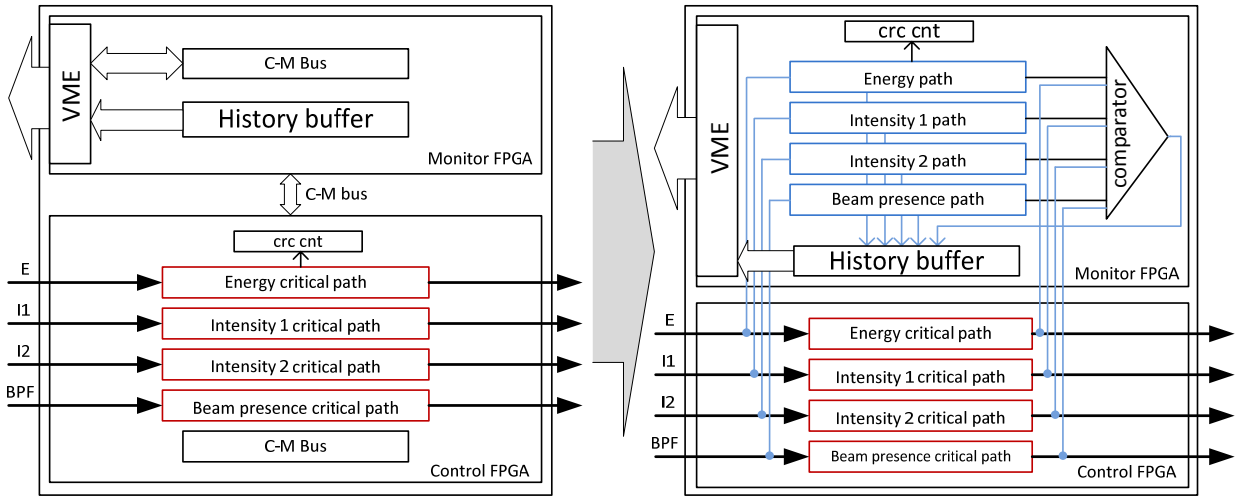

Figure 4: Old (left) vs. new (right) CISR implementation.

Monitoring the Critical Path

In the new implementation all non-critical elements were moved to the monitor device, and signal taps were routed to the monitor device to allow monitoring of the critical path operation.

In effect there is a *copy* of the critical function being executed in the monitor device, comparators ensure that the two devices give the same results, logging calculations steps and raising an alarm if problems arise.

It should be highlighted here that the failure of the on-line monitoring cannot have influence on the operation of the safety related logic and the critical data should still be received and retransmitted.

Lower Device Usage

The primary aim of the optimization was to simplify the design by re-partioning critical and non-critical function. Due to this re-arrangement, the control FPGA occupancy was significantly lowered as shown in Table 1 below. This has several benefits:

- Flexibility for future design requirements.
- Easier achievement of device timing constraints.
- More resources can be dedicated to the embedded logic probe, when it is used.

Table 1: Critical PLD Occupancy for Old vs. New Design

	Old	New	Change
Look-Up-Tables	63%	42%	**-21%**
Flip Flops	43%	35%	**-8%**
Memory	21%	0%	**-21%**

Simulation Test-Benches and Code Coverage

In addition to the simplification of the hardware, the aim was to maintain or improve the code-coverage of the design. The same test-bench and the same stimulus was applied to both the old and new behavioural designs, results are shown in Table 2 opposite.

In effect almost complete coverage of the modified areas of the control FPGA design was achieved, the areas yet to be modified result in total coverage falling somewhat short of 100%.

Table 2: Critical PLD Coverage for Old vs. New Design

	Old	New	Change
Statement	91.9%	92.9%	**+1.0%**
Branch	87.6%	91.7%	**+4.1%**
Condition	81.7%	85.6%	**+3.9%**
Expression	79.4%	81.8%	**+2.4%**

CONCLUSIONS

Methods and metrics to evaluate design improvements are in development, preliminary ideas have been discussed in this paper, with reference to a redesign of a sub-system of the LHC SMP controller.

In this case, further partitioning of the mission-critical functionality was possible, significantly simplifying the HDL design whilst preserving similar levels of code-coverage and the online monitoring capabilities.

The receiver design still needs further improvement, the approaches outlined here will be further enhanced, and applied to the rest of the components in the receiver module, and ultimately of the whole SMP system.

On-demand and stand alone testing is a key area, the hardware was only verified using a data generator to simulate working conditions, a dedicated tester is in development for future testing.

REFERENCES

[1] "International Standard IEC 61508-7", first edition 2000-03.

[2] David Dempster, Michael Stuart, "Verification Methodology Manual. Techniques for Verifying HDL Designs", ISBN 0-9538-4822-1.