Point Lepreau Refurbishment Project Programmable Digital Comparator (PDC) Replacement Software Unit and Integration Test

A. Xing[1], A. Condor[1], G. Raiskums[1], and D. Dickson[2]
[1]Atomic Energy of Canada Limited, Mississauga, Ontario, Canada
[2] NB Power Nuclear, Point Lepreau Generating Station, New Brunswick, Canada

Abstract

As part of the Point Lepreau Refurbishment Project the Programmable Digital Comparators (PDCs) for both shutdown systems are being replaced. The overall PDC development life cycle together with a progress update was presented in a paper submitted last year. One of the key activities in the PDC development lifecycle is software testing. This paper briefly describes the progress made on the PDC development since last update and focuses on PDC software testing. The project has completed the software requirements specification as well as the software design and these have been formally reviewed/verified. Software testing for both shutdown systems is underway. The software test process for SDS1 is described in this paper.

1. Introduction

Two completely independent shutdown systems, SDS1 and SDS2, are used in CANDU®[1] (CANada Deuterium Uranium) reactors. Each system contains three independent safety channels arranged in a 2-out-of-3 voting system. Channelized instrumentation is used to monitor a number of plant neutronic and process variables. If variables in any two channels of a single system are outside pre-determined envelopes, a shutdown is initiated.

At the Point Lepreau Generating Station (PLGS) the logic for most of the process-related reactor trip coverage is implemented in trip computers, commonly referred as Programmable Digital Comparators (PDCs). This allows optimization of trip functions for various operating conditions using power conditioning logic and power dependent setpoints, etc. The PDCs also implement self-checking and equipment monitoring functions to improve availability and decrease the maintenance and testing work load on the station staff.

As part of the Point Lepreau Refurbishment Project the PDCs for both shutdown systems are being replaced in order to ensure safe and reliable operation of the plant for a further 25-30 years.

The PDC platform selection/qualification and overall software development lifecycle are described in [1]. The adopted software lifecycle (shown in Figure 1) consists of a distinct set of activities that forms an integral part of the engineering of the PDCs. The intent of the software lifecycle is to achieve the required outputs through a comprehensive sequence of development, verification and validation steps. Software development for the PDCs consists of three phases,

[1] CANDU® is a registered trademark of Atomic Energy of Canada Limited (AECL).

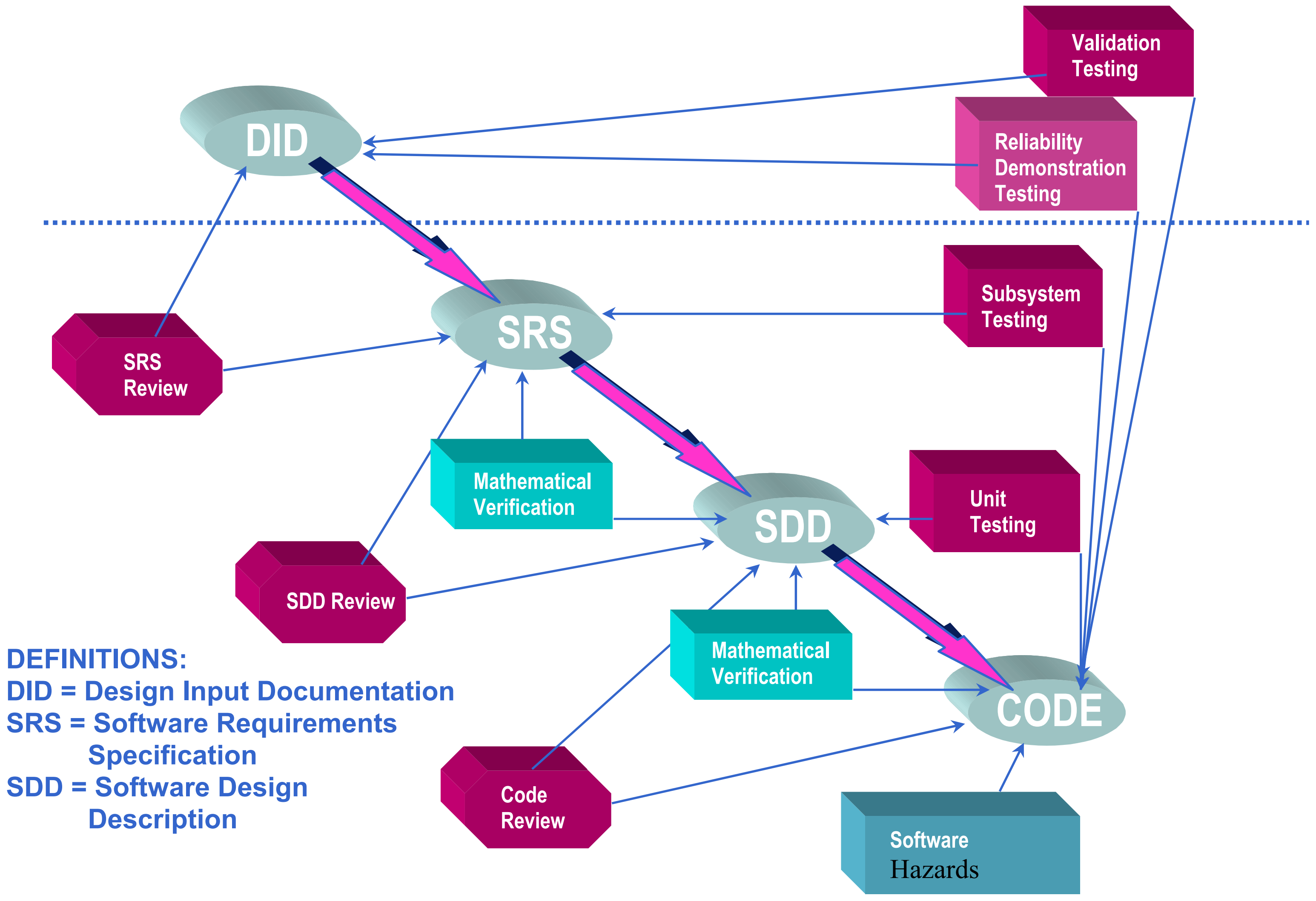
Validation Testing
DID
Reliability Demonstration Testing
Subsystem Testing
SRS Review
SRS
Mathematical Verification
SDD
Unit Testing
SDD Review
Mathematical Verification
CODE
Code Review
Software Hazards
DEFINITIONS:
DID = Design Input Documentation
SRS = Software Requirements Specification
SDD = Software Design Description

Figure 1 PDC Software Lifecycle

the software requirements specification (SRS) which is derived from the design input documentation (DID), the software design description (SDD), and coding.

Verification is performed to ensure that each phase of the design implements correctly the requirements in the previous phase. The verification activities consist of SRS review, SDD review and verification, code review and verification (for SDS2), and software testing. Unit and integration software testing are used to test the code against the SDD and SRS. Each verification activity is performed independently and rigorously following applicable procedures with findings documented in a review/verification report.

The software hazards analysis is used to identify potential hazards in the software design and code. This process can lead to changes that make the software more robust and resistant to failures. Finally, validation and reliability qualification check the PDCs against the DID requirements.

This paper provides an update on the progress of PDC replacement and focuses on one of the PDC software verification activities - PDC software test (for SDS1).

2. Progress update on PDC replacement

The project has undergone the second pass of the PDC software development and verification phase. The second pass was initiated to implement a number of design changes (i.e., added trip parameters). Two independent teams (each consisting of independent developers and verifiers) are dedicated to SDS1 and SDS2, respectively, and both are progressing at nearly the same pace for the software development and verification activities. The PDC replacement status for each activity in the PDC lifecycle (Figure 1) is summarized below.

- **Software Requirements Specification and Review**
  The SRSs for both SDS1 and SDS2 have been revised and formally issued. Formal reviews on the revised SRSs have been completed with review results documented in the revised SRS review reports.

- **Software Design Description, Design Review and Verification**
  The SDDs (for both SDS1 and SDS2) have been revised and formally issued. Formal SDD reviews and SDD verifications have been repeated with review/verification results documented in the revised SDD review reports and SDD verification reports, respectively. For SDS1, code is generated automatically from the SDD, which eliminates the manual coding stage. For SDS2, manual coding has been completed, which was followed by a formal code review and verification.

- **Hazard Analysis**
  The impact of the design changes (i.e., added trip parameters) on the hazards analysis reports (for SDS1 and SDS2) has been assessed and documented in the updated hazards analysis reports for SDS1 and SDS2, respectively.

- **Hardware Procurement**

The Maintenance And Diagnostic (MAD) systems for SDS1 and SDS2 were delivered to AECL in November 2006 and January 2007, respectively. The PDCs for SDS1 and SDS2 have passed the factory acceptance testing and were delivered to AECL in March 2007.

- **Software Test**
  PDC software test activities for SDS1 and SDS2 have started. PDC software test includes software unit test and software integration test.

The engineering of the PDCs for bothe SDS1 and SDS2 is on track with schedule and quality objectives. This paper will focus on one of the verification activities - software testing, and provide details on software test strategy for SDS1.

3. The SDS1 PDC application software overview

Different development processes are used for SDS1 and SDS2. SDS1 uses the Integrated Approach (IA), which is a formal methodology, developed by AECL for the design and verification of safety-critical software. It proceeds from the design input documentation through the entire software development cycle. The software requirements are specified in function block diagrams, which are further refined to produce the software design. A major characteristic of the IA is the use of a mathematically precise function block language to specify the software requirements and design, and to automatically generate executable code from the software design. The function block language used for PLR is defined in the IEC 61131-3 standard [2].

The application software is organized in a hierarchical presentation which optimizes the ease of software review by reviewers from different disciplines. The system overview diagram, at the top level of the hierarchical presentation, contains all the trip loops and the data flow links between them. Each trip parameter is typically implemented as a program at the intermediate level using the Function Block Diagram (FBD) language. A program may contain multiple sheets. A sheet within a program may implement input signal rationality check logic, signal correction logic, or the parameter trip logic. Commonly used logic, such as spread check logic, power conditioning logic, is implemented as user-defined function blocks (at the bottom level of the hierarchy) which are re-used in trip application software in the same way as the basic IEC 61131-3 blocks. After exhaustively tested and qualified, user-defined function blocks can be re-used for reliable software design by eliminating potential coding errors. User-defined function blocks can be part of the application software or can be contained in a user-defined library and exported for use in other applications.

4. Test of software implemented using FBD

The overall objectives of software testing are to find faults in the software and problems in the requirements. Software testing includes unit test and integration test which are described in the following sections. Unit test and integration test are intended to establish a high degree of confidence that the PDC software is error free and performs as intended. Software testing includes the following activities:

- defining a complete set of test cases following applicable software test procedure,
- documenting the test cases in the Unit and Integration test plan,
- conducting unit and integration testing (on the target hardware with pre-developed software), and
- documenting the test results in the testing report.

For applications implemented in FBD, unit and integration testing (described in Section 5 and Section 6, respectively) can be combined because of the following reasons.

- the unit test cases are executed in the same environment as the integration test cases (i.e., both unit test and integration test are performed on the actual PDC hardware platform),
- both SDD and SRS are specified using function block diagrams and the structure of the SDD diagrams is similar to the structure of the SRS functions (there is not a significant increase in complexity or added functionality between the SRS and SDD) and hence, the limited possibility of further errors does not warrant a separate layer of testing,
- program sequence control logic is encapsulated within function blocks that are exhaustively tested and qualified as part of the platform qualification; testing all possible condition/decision outcomes (in unit testing) is a subset of testing each functional requirement (in integration testing).

Given the repetitive and time consuming nature of test execution, testing of the PDC is automated as much as possible using a custom test facility and scripting tool. Manual tests are performed in cases where it is not suitable to automate, as in some integration tests.. Automated test cases cover unit tests, integration tests and response time tests. Manual test cases include functional timing tests and selected response time tests, etc.

5. PDC software unit test

The objectives of unit testing are:

- to find faults in the translation from source code to executable code,
- to test that the executable code of each program/module behaves as specified in the SDD,
- to test that the executable code of each program/module does not perform unintended functions, and
- to find faults in the program interfaces.

5.1 Unit test methodology

For application implemented in FBD, each page (sheet) of diagram or each user defined function block is considered a unit. A sample unit (a page of diagram) is shown in Figure 2.

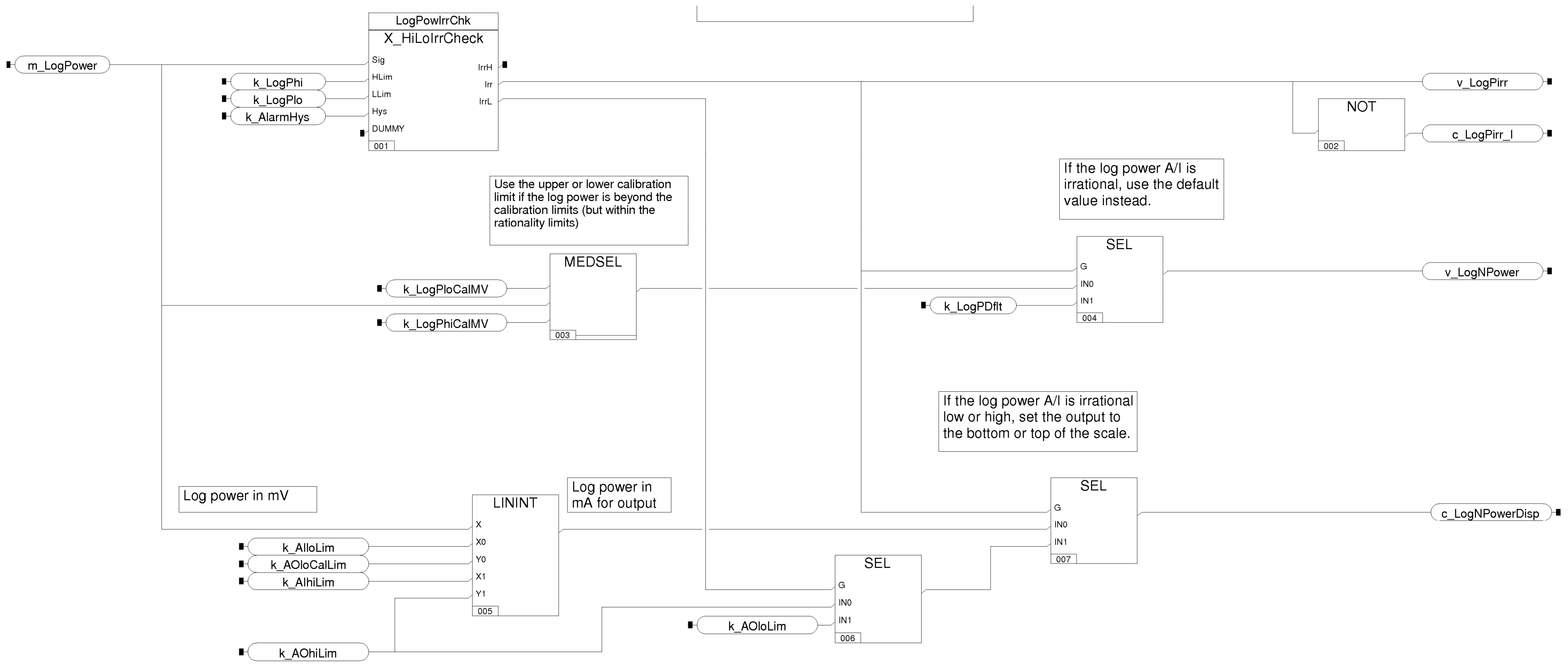
m_LogPower
LogPowIrrChk
X_HiLoIrrCheck
Sig
HLim
LLim
Hys
DUMMY
IrrH
Irr
IrrL
001
k_LogPhi
k_LogPlo
k_AlarmHys
v_LogPirr
NOT
002
c_LogPirr_I
If the log power A/I is irrational, use the default value instead.
Use the upper or lower calibration limit if the log power is beyond the calibration limits (but within the rationality limits)
MEDSEL
003
k_LogPloCalMV
k_LogPhiCalMV
SEL
G
IN0
IN1
004
k_LogPDflt
v_LogNPower
If the log power A/I is irrational low or high, set the output to the bottom or top of the scale.
Log power in mV
LININT
X
X0
Y0
X1
Y1
005
Log power in mA for output
k_AIloLim
k_AOloCalLim
k_AIhiLim
k_AOhiLim
SEL
G
IN0
IN1
006
k_AOloLim
SEL
G
IN0
IN1
007
c_LogNPowerDisp

Figure 2 A Sample Software Unit (in Function Block Diagram)

The following characteristics of the PDC platform facilitate the use of intrusive test for unit testing.

- Any internal variables can be read during unit test without disturbing the software logic,
- Any internal or input variable can be written with test values during testing by making trivial changes to the software that have no effect on program sequence, program logic or other side-effects.

Note that this intrusive testing mode is necessary to test any functions that cannot be tested via monitored variables.

The test PC is connected to the target via Ethernet link and the test tool communicate with the target using the DDE/OPC communication protocols (Figure 3).

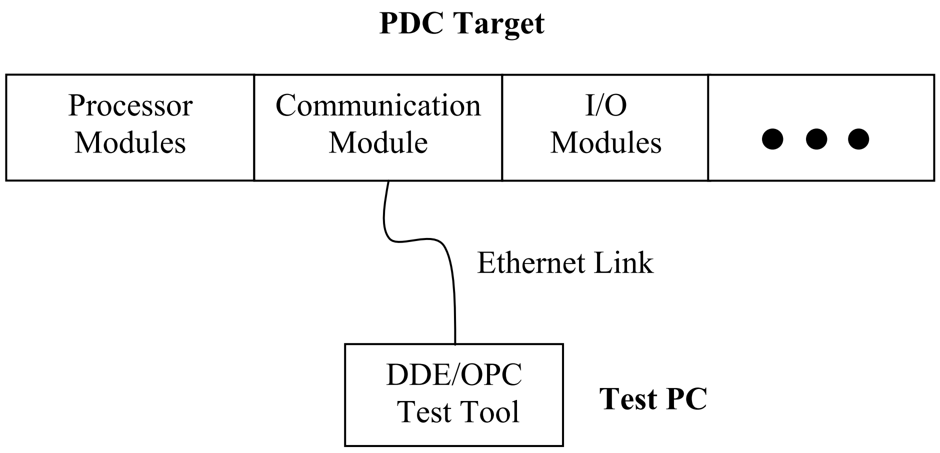

Figure 3 Unit Test Configuration

The intrusive unit test is automated using an in-house developed test tool. Test cases are defined in ASCII text files (also referred to as test scripts) and are executed automatically by the test tool running on the test PC.

5.2 Unit test cases definition

Test cases are selected to provide the following coverage requirements.

- all possible decision outcomes,
- tests on each boundary and values on each side of each boundary for each input (the test values are determined using boundary value analysis and equivalence partitioning),
- tests based on postulated coding implementation errors.

In FBD language, decision outcomes are typically implemented using selection blocks (e.g., SEL and MUX as shown in Figure 4) whose output depends on the value of a decision variable (either a Boolean variable G (= false or true) as in the case of SEL or an integer variable K (= 0, 1, …, n-1) as in the case of MUX). Testing all possible decision outcomes requires exercising all corresponding values of the decision variable.

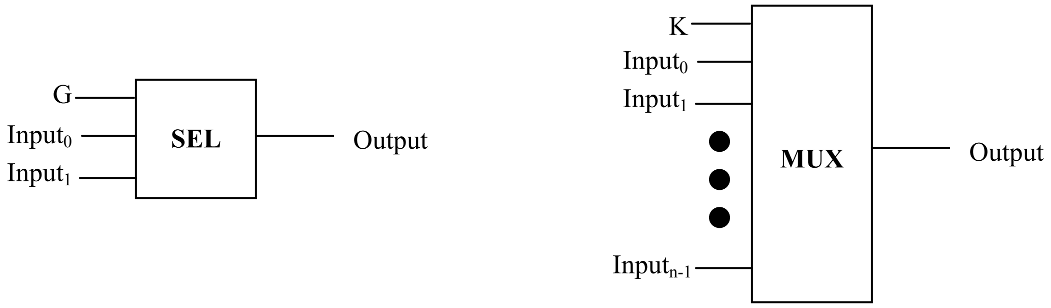

Figure 4 Example Selection Blocks

The domain of an analog input variable is typically divided into several regions (e.g., rational, irrational low and irrational high regions) as a result of equivalence partitioning. Boundary value analysis can then be applied to select input test values (typically at their minimum, just above the minimum, a nominal value, just below their maximum and at their maximum).

In general, each test case specifies one or more input values to be written to the target and the expected output values. For the example function block diagram unit shown in Figure 2, a sample unit test script is presented in Figure 5.

The majority of test cases are defined in the test script using the "TEST_TABLE … END_TABLE" construct. As shown in Figure 5, the first line defines the variable names for read and write. Variables listed after SET (and before TEST) are input variables to be written with specified input values; variables listed after TEST are those to be tested. Test cases are listed in subsequent lines. The first column is the test case numbers and the rest columns are the input values and expected output values corresponding to each variable name specified in the first line.

When a test script file is executed or interpreted by the test tool, each input value is written to the target by the test tool; the corresponding output values are read back from the target and compared with the expected values specified in the test scripts to determine test case failure or success. The test tool saves all test results in a test log file. To increase test capability and test cases reviewability, the test tool also supports other commands such as WAIT, DEFINE, etc.

Unit test cases based on the latest SDD have been fully developed following the applicable test procedure and documented in the test plan. Unit test is performed on the target PDC and test results will be documented in the test report.

```
# Log Power

# Conditions tested:
#       - rational (and greater than LCalibLim)
#       - irrational high
#       - rational and  <= LCalibLim
#       - irrational and <= LCalibLim
#       - irrational low
# Notes:
#       - HCalib limit corresponds to irrational HLim, however the LCalib limit does
#         not correspond to the irrational LLim
#
#       o_LogNPowerDisp = (4095-819)/(5000-0) * (v_LogPowerAI - 0 ) + 819 (if rational)
#
#       o_LogNPowerDisp = k_AOhiLim (4095, if irrational high)
#       o_LogNPowerDisp = k_AOloLim (0, if irrational low)
#
#
define HLim 4588.0
define LLim 200.0
define Default 4520.7
define LCalibLim 1000.0
define Hys 50.0
define M 3.5
#
TEST_TABLE
SET    m_LogPower    TEST o_LogPirr_I   v_LogPirr       v_LogNPower       o_LogNPowerDisp
1          2500.0           1             0                2500.0               2457
2          HLim-M           1             0                HLim-M               3823
3          HLim+M           0             1                Default              4095
4          5000.0           0             1                Default              4095
5          5300.0           0             1                Default              4095
6          HLim             0             1                Default              4095
7          HLim-Hys+M       0             1                Default              4095
8          HLim-Hys-M       1             0                HLim-Hys-M           3790
9          LCalibLim+50.0   1             0                LCalibLim+50.0       1507
10         LCalibLim-50.0   1             0                LCalibLim            1441
11         LLim+M           1             0                LCalibLim            952
12         LLim-M           0             1                Default              0
13         0.0              0             1                Default              0
14         LLim             0             1                Default              0
15         LLim+Hys-M       0             1                Default              0
16         LLim+Hys+M       1             0                LCalibLim            985
17         LCalibLim-50.0   1             0                LCalibLim            1441
18         LCalibLim        1             0                LCalibLim            1474
19         LCalibLim+50.0   1             0                LCalibLim+50.0       1507
END_TABLE
```

Figure 5 A Sample Unit Test Script

6. Integration test

The objectives of the integration testing are:

- to test that the software modules integrated together and with the target hardware and pre-developed software meets the requirements specified in the SRS,
- to find errors in the software, hardware, and pre-developed software interfaces, and
- to find errors in handling stress conditions, timing, fail-safe features, error conditions, and error recovery

6.1 Integration test methodology

Integration test is performed at the I/O electrical level (i.e., non-intrusive testing) using a test facility. The PDC software is integrated with the PDC hardware, system software and I/O sub system, and the PDC software is tested on the PDC target without modifications. The test facility has its own CPU and is connected to the PDC target via I/O terminal blocks. The test facility writes values to the PDC monitored variables via its output modules and reads PDC controlled variables from its input modules (Figure 6).

Depending on the capability of the test facility, integration test cases can be either developed and executed directly on the test facility processor or performed by an external test PC connected to the test facility as in the case for unit test. For PLR SDS1 PDCs, integration test is performed using an external test PC.

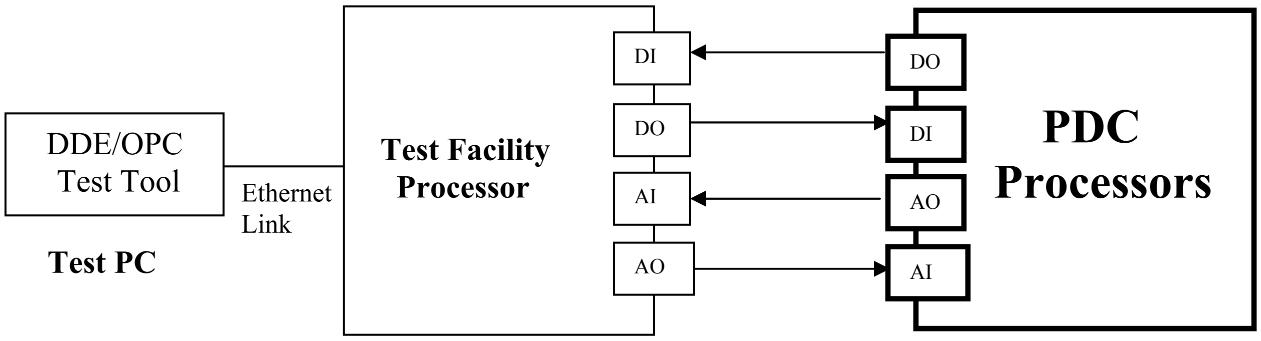

Figure 6 Integration Test Configuration

6.2 Integration test cases definition

Integration test cases are selected to provide the following coverage requirements

- Define test cases to test each functional requirement in the SRS.
- Define test cases to test the performance requirements specified in the SRS.
- Identify all resources used by the subsystem software and define test cases which test the subsystem under conditions that attempt to overload these resources in order to determine if the functional and performance requirements defined in the SRS are met.
- Define test cases to exercise any interfaces between:
    - The software and the target hardware,
    - The software and pre-developed software,
    - The software modules themselves,
- Define test cases to show that the subsystem meets its requirements under each hardware configuration and operation option.
- Define test cases to test the ability of the subsystem to respond as indicated in the SRS to software, hardware and data errors.

- Define test cases which attempt to subvert any existing safety or security mechanisms.

Integration test scripts are defined in the same scripting language as in unit testing and are executed using the same test tool. Integration test scripts have been developed and referenced in the test plan. Integration test is performed using the test facility and test results will be documented in the test report.

7. Test tools and facilities

To automate PDC application software testing, a test script interpreter has been developed using Visual Basic. The scripting tool executes test scripts and logs test results automatically. It also displays input values written to the target and output values read from the target. Test scripts can be executed separately or can be contained in a single file to be executed one after another automatically. The scripting tool communicates with the target using DDE/OPC protocols. Any global variables with the READ or WRITE attribute in the target can be accessed by the test tool using DDE/OPC. This intrusive test method is used for unit test of the PLR PDC software.

A PLC has been selected as the test facility for integration test. Test cases are applied via signals at the I/O terminal block. The selected test facility supports communication with external systems through a standard communication protocol (e.g., DDE/OPC).

8. Concluding remarks

The PLR SDS1 PDCs use the Integrated Approach (IA), which is a formal methodology developed by AECL for the design and verification of safety-critical software. A major characteristic of the IA is the use of a graphical programming language. The function block language used for PLR is defined in the IEC 61131-3 standard. Testing of software implemented in FBD language is automated to a great extent. To support automated software test, a customized test scripting tool and test facility have been developed. PDC software test activities for SDS1 and SDS2 have started and are expected to finish within a month. The engineering of the SDS1/SDS2 PDCs is on track with schedule and quality objectives.

9. References

[1] Fraser, K.G., et al., “Point Lepreau refurbishment project programmable digital comparator (PDC) replacement for SDS1 and SDS2 – update 2: software development and review and validation test rig development,” 27th Annual Conference of the Canadian Nuclear Society, Toronto, June 2006.

[2] IEC 61131-3, “Programmable Controllers – Part 3: Programming Languages”, First Edition, 1993.