Smart Reconfigurable Test Software Design Description

Author: **Sean McMahon**

Department: **Software**

Contributors: **Sarah Brady**

Most Marvellous Company Ltd.

9 Inchicore Terrace North, Inchicore, Dublin 8, Ireland

WW.MOSTMARV.COM

1 Data Item Description

1.1 Revision History

Version	Status	Date	Author	Change Description
0.1	Draft	13/11/2019	S McMahon	Initial draft
0.2	Draft			Document Review
1.0	Release			

1.2 Purpose

Introduce the project. Describe the SR System Architecture.

1.3 Stakeholders

Stakeholder should indicate their approval.

Position	Name	Date
Project Owner	Garret O'Doherty	

2 Introduction

The Smart Reconfigurable System (SRS) project aims to design and implement a reconfigurable system that can be deployed in multiple environments such as:

- Automotive Function Test
- Semiconductor Evaluation
- Medical Device Validation

There is no universally recognised off the shelf solution that caters to the above manufacturing environments and as a consequence, companies in these sectors tend to design their own manufacturing systems. This places a significant financial burden on companies as a dedicated, skilled and costly team must be on hand to run maintain their systems. There is typically a steep learning curve for new entrants as they attempt to familiarise themselves with a system that may be decades old and build upon obsolete software frameworks.

The goal of this section of the SRS project is as follows:

- Select an existing robust and scalable architecture.
- Propose a reconfigurable architecture.
- Implement the architecture in a demonstration project.
- Design an attractive GUI.

3 Architecture

The language chosen for the project is LAbVIEW. In LabVIEW, there are many well established, such as:

- Simple State Machine
- Queued Message Handler
- LabVIEW Component Object Design (LCOD)
- Actor Framework

Actor Framework is the framework most promoted by National Instruments), the progenitor of LabVIEW), however, it is a difficult and daunting framework to follow for most mid-level LabVIEW developers.

With this concern in mind LabVIEW Component Object Design (LCOD) was selected. LCOD was introduced by Stew Watts & Jon Hall in their 2003 book, “A Software Engineering Approach to LabVIEW™”.

LCOD is an approachable framework that used lvlibs to implement components which can be acted upon. The architecture has much in common with object orientated analysis but is a different system defined by its cohesion, coupling and information hiding. Essentially, LCOD defines components and then defines an API that can communicate with the component.

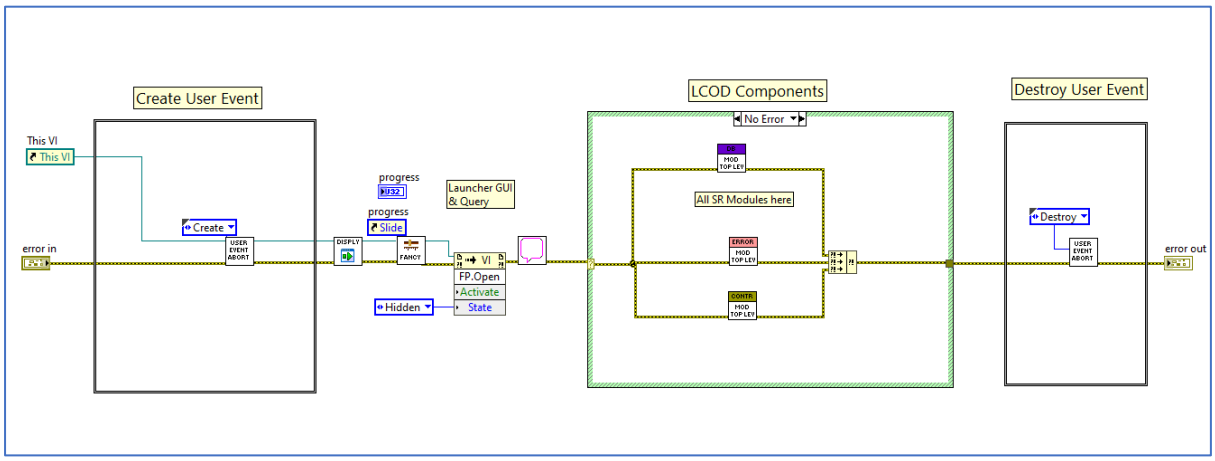


Figure 1 Basic LCOD Architecture (SR Test Launcher.VI)

The launcher is very simple and merely sets up a user event and then launches the components. The user event can be used to tear down on the components in the desired manner and in the interim the components run independently but with an API structure and event structure that allows inter-process communication and control.

The LCOD components themselves use the familiar queued message handler design pattern as illustrated in Figure 2

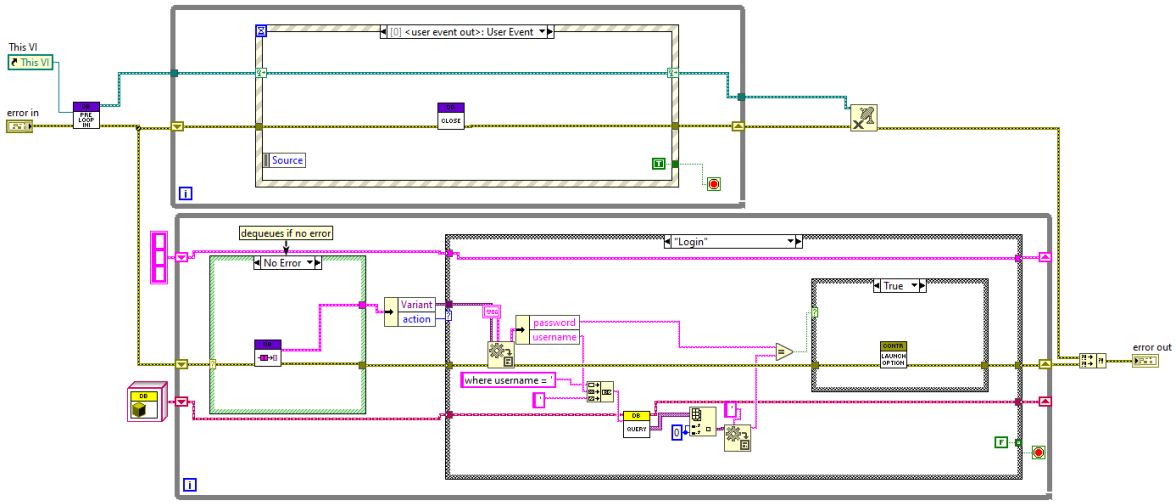


Figure 2 Queued message Handler design pattern in the database component (Database Module.vi)

Typically, projects using LCOD will have components such as:

- Controller
- GUI
- Error Handler
- Database Handler

4 Proposed Smart Reconfigurable System

Having selected LCOD as the design architecture. The next goal was to make it configurable so that large chunks of functionality could be encapsulated in a standardised framework.

The block diagram in Figure 3 illustrates how a base system could be designed and then reconfigured for a particular project/ application/

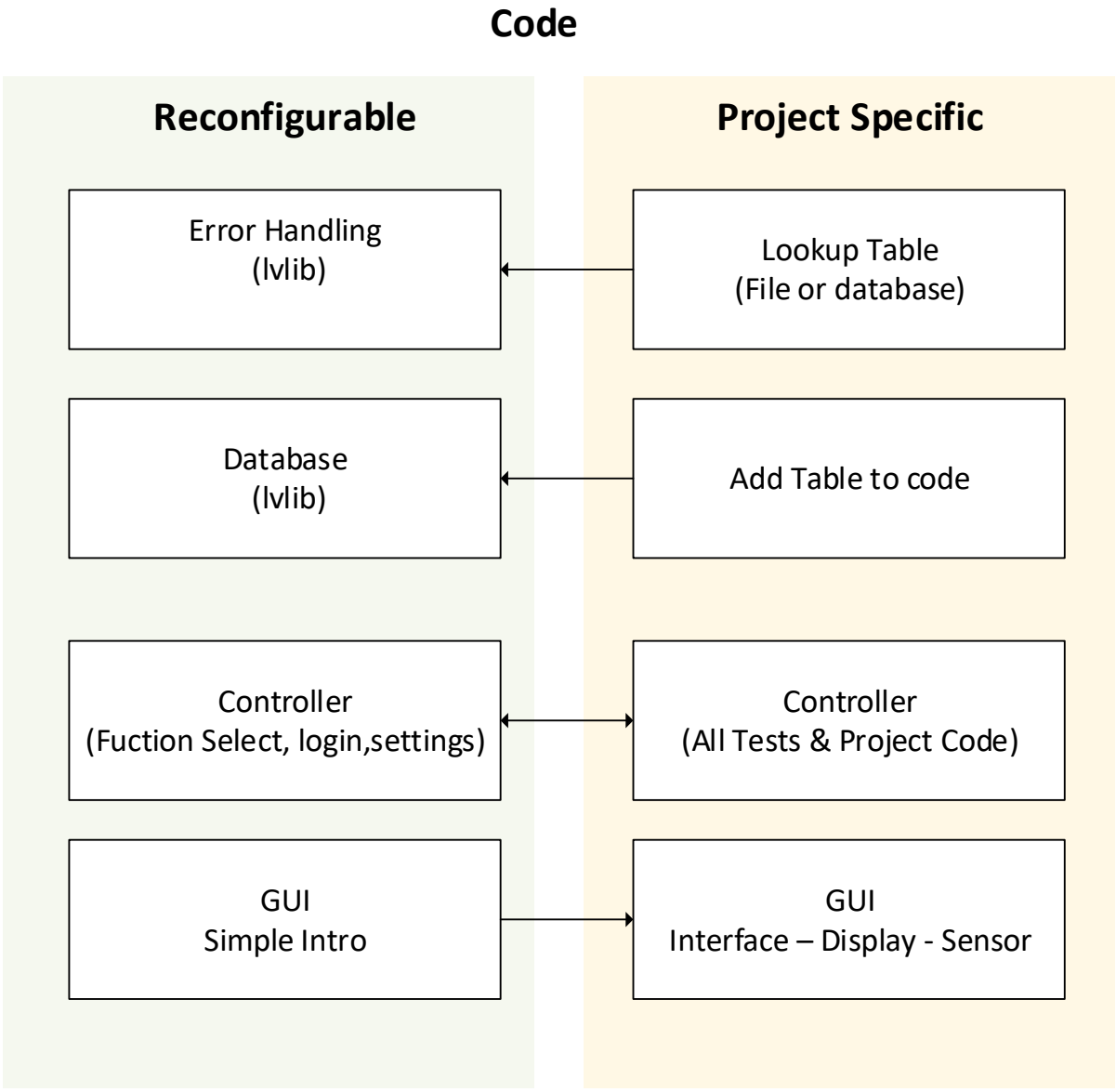


Figure 3 Smart Reconfigurable System

5 Implementation

The proposed architecture has been implemented in the LabVIEW project: SR Test.lvproj

The demonstration system has a base controller component and database component. The controller component queries the database component for a username and password. If correct the specific system is launched, thus illustrating the reconfigurable potential.

6 Graphical User Interface (GUI)

LCOD a design pattern that is an established, stable and robust pattern. However, the code is often something with which a customer does not trouble themselves assuming the system works. In the 21st century, it is now important to consider human factors and especially when attempting to win customers.

The GUI is design using the DMC GUI palette and is designed to be informative, intuitive and pleasant to view. It has a modern feel that assists in winning customers.

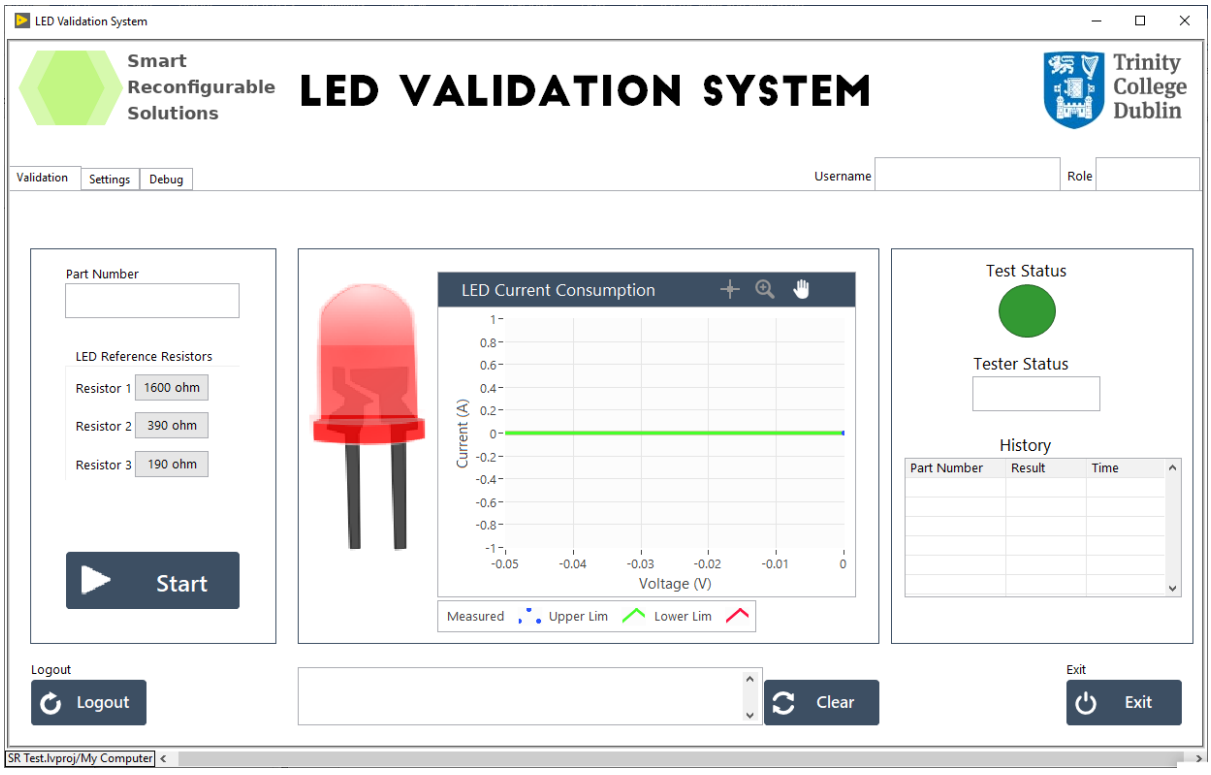
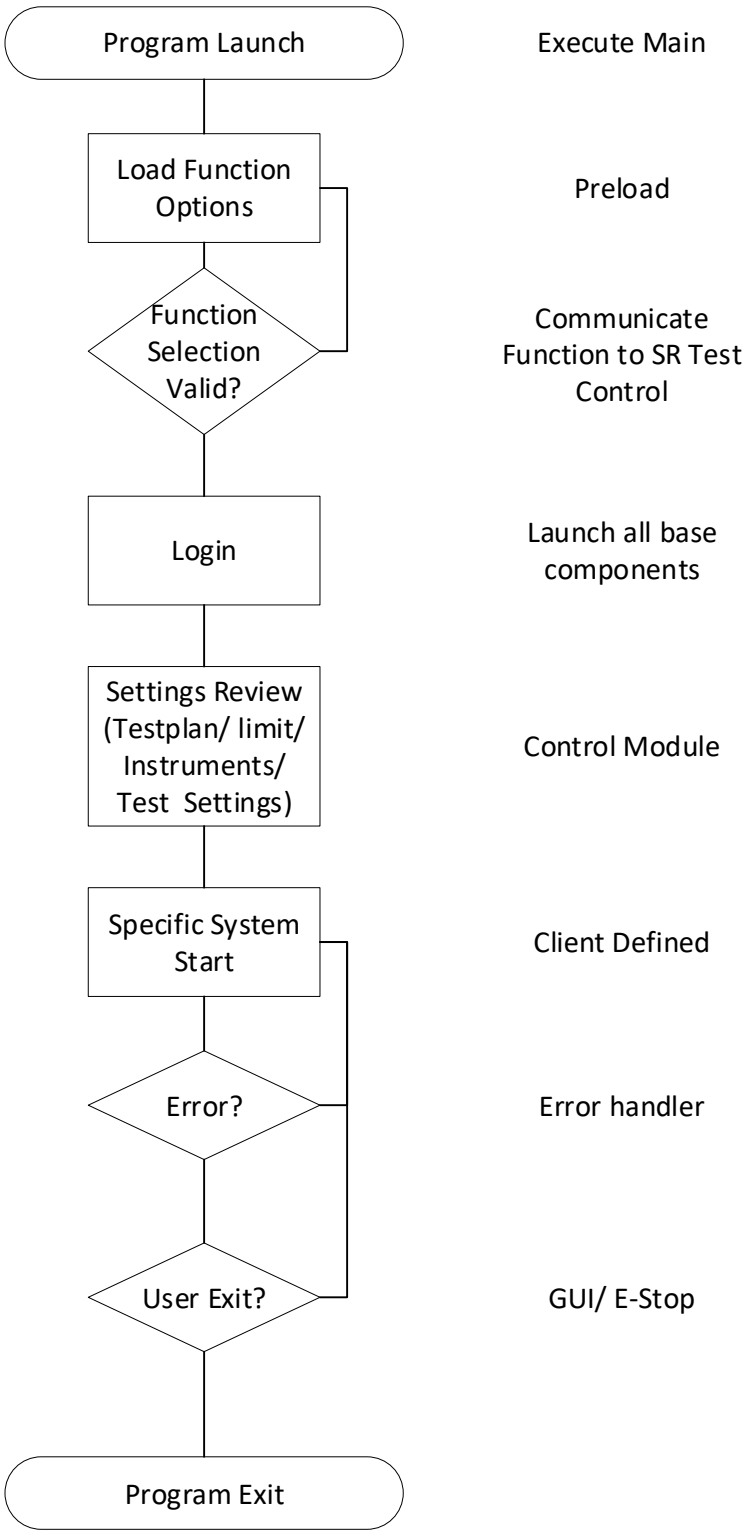


Figure 4 GUI

7 Appendix

7.1 SR Test Program Flow



7.2 Code rationale

7.2.1 Load Function

Option A

INI files.

Advantages : Users cannot arbitrarily add sections to the INI file, (LabVIEW required)

Disadvantages: Users cannot arbitrarily add sections to the INI file, (LabVIEW required)

Option B

Text files and Sorter

Advantages : Users can arbitrarily add sections to the INI file, (LabVIEW not required)

Disadvantages : Users can arbitrarily add sections to the INI file, (LabVIEW not required).

Decision

Text file will point to a folder in which the configurable sections are contained. Easy to add in new functions without the need for LabVIEW on an executable system.

7.2.1 Settings Review

INI files.

Advantages : Users cannot arbitrarily add sections to the INI file, (LabVIEW required)

Disadvantages: Users cannot arbitrarily add sections to the INI file, (LabVIEW required)

Option B

Text files and Sorter

Advantages : Users can arbitrarily add sections to the INI file, (LabVIEW not required).

Disadvantages : Users can arbitrarily add sections to the INI file, (LabVIEW not required).
Complexity increases

Decision: Option A. Time is limited and considerable thought and development required to investigate benefit of configurability of settings via text files. System is still reconfigurable but LabVIEW, Source Code and deployment capability required. Essentially, provider will redeploy.