Missile Defence System - ADLAS

Written by Simon Crowther

# Abstract

The Main aspect of this paper is to create, develop and test a missile defence system that will target incoming projectiles. Systems of this nature are often complex and perform high levels of algorithms to calculate correct values for ensuring absolute best performance. Moreover, designing a system of this nature will allow a certain level of understanding on how to map a system to requirements and ensure the re-usability.

The end goal of this project will enhance the understanding of how the system will cope with unexpected scenario’s and will affect the performance, once the system has been tested and then each possible scenario compared such as distance, altitude and angle of trajectory. All quality and event variables will have collected to be up and hence analysed to find a measure of performance that is useful in assessing the validity of the system tested and also to seek a final evaluation of what is happening.

**Keywords = Ada, Requirements, Traceability, V software lifecycle**

Contents

[Abstract 2](#_Toc494737256)

[Proteus Design 6](#_Toc494737257)

[Problems solved so far 8](#_Toc494737258)

[What I have learnt from this Project? 14](#_Toc494737259)

[Conclusion 14](#_Toc494737260)

[Appendix 16](#_Toc494737261)

# List of Abbreviations

|  |  |
| --- | --- |
| Abbreviation | Description |
| BS | Base Station |
| FPT | Flight Phase Tests |
| HLD | High Level Design |
| LLD | Low Level Design |
| LPT | Launch Phase Tests |
| IPT | Initialised Phase Tests |
| SDLC | Software Development Life Cycle |
| SPT | Simulation Phase Tests |
|  |  |

# List of Figures

Figure 1: STM32 On Begin - Page 15

# 1 Introduction

## 1.1 Background

For this project, it was the idea to focus on an idea that has become very important in today's defense society. It has become apparent that missiles can be fired from any point of the earth and with the speed capabilities and high probability of not being able to destroy the object before it hits its intended target. I wanted to focus on something I found fascinating and to really look at how complex of a system of this nature would be and if it was capable of being miniaturized onto a STM32F4 board and coded in Ada 2012 which is a strongly typed language.

This system needs to calculate what values bout the other object is necessary to calculate the probability of nullifying the object and whether it would be possible within the timeframe of detecting to endpoint. Throughout this project, I will look at various software techniques used to work out the most efficient way of software development and if it’s possible to ensure that software quality can be maintained throughout the project.

## 

## 1.2 Aim

The overall focus of this project is to set requirements, design, implement and code a complex missile defense system.

## 1.3 Objectives

* Review theoretical aspects of software development
* Review components of missiles and counter measures
* Review ADA software
* Set requirements for the system
* Design and code the system
* Provide verification of software meeting requirements
* Perform system and unit testing
* Produce conclusions on the project

## 1.4 Design Points of Objectives

This section is to explain each objective and how it will be reached, it will then be reviewed in the last chapter Conclusion to see if it was completed.

* Review theoretical aspects of software development – Look at different SDLC models and which will fit this project suitably, understand how each one has its advantages and disadvantages.
* Review components of missiles and counter measures – Further review into physical aspects of missiles and how they operate, what measures can be used to counter and how current systems calculate choices.
* Review ADA software – Researching how Ada can be used to be safety and mission critical and aid the user to falling into similar coding pitfalls.
* Set requirements for the system –Create requirements for what the system should operate to and ensure that traceability of requirements to code is followed throughout.
* Design and code the system – Create the system required for main functionality to match the specification
* Provide verification of software meeting requirements – Ensuring that the code is fulfilling what it was designed to do.
* Perform system and unit testing – Ensuring the values and system performs the expected results
* Produce Conclusions for the project – Explain the positives and negatives for the project and whether the end product matched the design set out.

## 1.5 System Overview

A circuit board

Description generated with high confidence

## 

The ADLAS and other related platform equipment function together to:

* Control and monitor the release and system critical information
* Identify, locate and quantify the target
* Report to the Mission System (MS) the simulated time of impact and overall coverage affectability
* Preparing ADLAS for Release
* Prepare suspension and release equipment for release of secondary payload
* Perform Emergency detonation

## 1.6 Dependencies

GD will provide an AdaCore Target Build environment. This will enable Resource Group to compile their code against the target minimal Ada run-time we are using.

An AdaCore Native Build environment will enable this system to be compiled and allow the code to be used for testing purposes such as Unit Testing.

In order to continue to maintain the signal to register bit mapping in a single place, it is expected that the Release DevBase packages should use the Signal\_To\_Register package to determine the correct register and bit associated with each signal.

The ADLAS Packages listed below:

* IPT
* SPT
* LPT
* FPT
* Common\_Checks

The different phases of ADLAS tests will be scheduled by the Schedule package. This Scheduler package will decide which Phase routines to call as well as when to call them based on the information in the ADLAS Definition Document. The ADLAS Test package will retrieve a copy of all the appropriate registers from the Inputs at the start of the frame to ensure a consistent set of input values for all the tests.

Each ADLAS Test will perform its required check, checking the appropriate register bits, and reporting any failure by calling the failure\_log-log\_failure routine with a Fault Data Record. The content of the fields of the Fault Data Record is defined in the ADLAS definition document.

## 1.8 IPT

This static class controls the execution of the ADLAS checks required to be performed during the First phase.

The IPT package shall provide the following operations:

### 1.8.1 Procedure Static\_Initialise;

The Static\_Initialise routine shall initialise all internal state of the class.

### 1.8.2 Procedure Schedule\_Phase;

The Schedule\_ Phase routine shall perform all the tests defined as IPT Tests in the ADLAS Definition Document. Once the complete set of tests has been performed, the sequence allows passage to the next phase or stops the function.

## 1.9 SPT

This static class controls the execution of the ADLAS checks required to be performed during the Second phase.

The SPT package shall provide the following operations:

### 1.9.1 Procedure Static\_Initialise;

The Static\_Initialise routine shall initialise all internal state of the class.

### 1.9.2 Procedure Schedule\_Phase;

The Schedule\_ Phase routine shall perform all the tests defined as SPT Tests in the ADLAS Definition Document. Once the complete set of tests has been performed, the sequence allows passage to the next phase or stops the function.

## 1.10 LPT

This static class controls the execution of the ADLAS checks required to be performed during the Third phase.

The LPT package shall provide the following operations:

### 1.10.1 Procedure Static\_Initialise;

The Static\_Initialise routine shall initialise all internal state of the class.

### 1.10.2 Procedure Schedule\_Phase;

The Schedule\_ Phase routine shall perform all the tests defined as LPT Tests in the ADLAS Definition Document. Once the complete set of tests has been performed, the sequence allows passage to the next phase or stops the function.

## 1.11 FPT

This static class controls the execution of the ADLAS checks required to be performed during the Fourth phase.

The FPT package shall provide the following operations:

### 1.11.1 Procedure Static\_Initialise;

The Static\_Initialise routine shall initialise all internal state of the class.

### 1.11.2 Procedure Schedule\_Phase;

The Schedule\_Phase routine shall perform all the tests defined as FPT Tests in the ADLAS Definition Document. Once the complete set of tests has been performed, the sequence allows passage to the next phase or stops the function.

## 1.12 ADLAS\_Tests

This static class implements the full suite of ADLAS tests as defined in the ADLAS Definition Document and makes them available for use by the ADLAS Scheduler objects.

In addition to the individual test procedures, the ADLAS\_Tests package shall provide the following interface:

### Procedure Static\_Initialise;

The Static\_Initialise routine shall initialise all internal state of the class.

### Procedure Read\_And\_Buffer\_Input\_Signals;

The Read\_And\_Buffer\_Input\_Signals routine will be called by the Scheduler at the start of the frame and shall read and store a copy of all the necessary registers in order to provide a consistent set of input values for all the tests.

## 1.11 Verification Activities

I shall perform the following Verification Activities on the Ada packages I develop and provide a test report for each:

* Package

## 1.12 Package Requirements

## General Requirements

|  | Software shall comprise the following Phase Modes:  Initialised Phase Tests  Simulation Phase Tests  Launch Phase Tests  Flight Phase Tests |
| --- | --- |

## IPT Requirements

|  | IPT shall be performed automatically and without any intervention of the user as soon as the Power-Up Manager Application passes the control to the Control Monitor. |
| --- | --- |
|  | IPT shall test Holding and Locking inputs for their expected default and self-consistent states. |
|  | Whenever IPT detects a failure the relevant fault shall be logged calling the Failure\_Log operations. |
|  | IPT shall complete within 3 minutes from Control passover. |

## SPT Requirements

|  | IPT shall be performed automatically and without any intervention of the user as soon as the IPT pass the minumum required locking safety checks. |
| --- | --- |
|  | The SPT shall perform non-intrusive testing of the expected flight path and relevant issues of the launch release, emergency misfire and emergency jettison hardware interface. |
|  | SPT shall be run under guaranted time frame which is to be agreed upon flight path needs. |
|  | Whenever SPT detects a possible failure the relevant fault shall be logged calling the Failure\_Log operations. |

## LPT Requirements

|  | LPT shall not run unless IPT and SPT pass over control. |
| --- | --- |
|  | LPT shall complete within given timeframe. |
|  | Whenever LPT detects a failure the relevant fault shall be logged calling the Failure\_Log operations. |
|  | LPT will run when access to fire has been given and all other checks are at the given threshold. |

## FPT Requirements

|  | FPT shall not run unless LPT passes the main sequence events. |
| --- | --- |
|  | FPT shall complete within the given SPT timeframe. |
|  | Whenever FPT detects a failure the relevant fault shall be logged calling the Failure\_Log operations. |
|  | FPT will run when LPT passes over control. |

# 2.0 System Design Packages

## 2.1 Section A

|  |  |  |  |
| --- | --- | --- | --- |
| Group | Group ID Offset | Max Test Id’s | Test ID Range |
| IPT | 0 | 999 | 0-999 |
| SPT | 1000 | 999 | 1000-1999 |
| LPT | 2000 | 999 | 2000-2999 |
| FPT | 3000 | 999 | 3000-3999 |

## 2.2 Section B

|  |  |
| --- | --- |
| Group | Procedures |
| IPT | Initialise  Schedule\_Phase  Run  Execute\_Light  Temperature\_Validation |
| SPT | Initialise  Schedule\_Phase  Run  Execute\_Light  Temperature\_Validation |
| LPT | Initialise  Schedule\_Phase  Run  Execute\_Light  Temperature\_Validation |
| FPT | Initialise  Schedule\_Phase  Run  Execute\_Light  Temperature\_Validation |