

STeP-IN SUMMIT 2007

International Conference On Software Testing

An Automated Testing Approach for Avionics systems

by

Shyam Sundar

Honeywell Technology Solutions Lab

Copyright: STeP-IN Forum and Quality Solutions for Information Technology Pvt. Ltd.

Published with permission for restricted use in STeP-IN SUMMIT 2007 in agreement with full copyrights from owner(s) / author(s) of material. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior consent of the owner(s) / author(s). This edition is manufactured in India and is authorized for distribution only during STeP-IN SUMMIT 2007 as per the applicable conditions.

Practices Experience Knowledge Automation

Produced By

STeP-IN
Forum

www.stepinforum.org

Hosted By



www.qsitglobal.com

1.1 Abstract

The importance of embedded software in avionics systems are ever increasing. Modern commercial aircrafts with Flight Management System and auto-pilots can fly the aircraft without the pilot's active intervention. These embedded softwares require full proof safety and high reliability. In these systems failure is simply no option as the results are catastrophic.

In order to ensure this kind of reliability aviation authorities across the globe has mandated a software development standard to be followed before the software can be deployed in the civilian aircraft. It is more popularly know as DO-178 B guidelines which specify final levels of criticality for any airborne software based on the impact of software malfunction impact. The avionics software needs to be developed as per the guidelines and recommendations given by DO-178B guidelines.

SW Level		Failure Conditions
a. <u>Level A</u>	←	a. <u>Catastrophic:</u>
b. <u>Level B</u>	←	b. <u>Hazardous/Severe-Major:</u>
c. <u>Level C</u>	←	c. <u>Major:</u>
d. <u>Level D</u>	←	d. <u>Minor:</u>
e. <u>Level E:</u>	←	e. <u>No effect</u>

There is enormous amount of effort involved in testing the software. As per the historical data and program metrics, it is predicted that almost 35% of the effort is spent on verification and validation activities. There are many initiatives to reduce the effort spent on verification without compromising on quality. One Method to achieve this through automation during the verification phase of the project. Identifying the parts of the project that follows a pattern and automating the verification will reduce the verification effort to a large extent.

This presentation we will be presenting will be one of the Initiatives that we have taken to automate the manual testing process in a product line "Flight Management System" (FMS) for domestic aircrafts. This Initiative can be carried across companies to drastically reduce the Verification cost without compromising on Quality.

1.2 INTRODUCTION

Stringent guidelines from the certification agencies to ensure safety have resulted in enormously high Testing effort. Testing effort plays a major role in deciding the cost of the product and deadline. There are instances where the product has not been delivered on time due to increase in testing time and effort. Automation of testing process is one of the Key methods to ensure safety and reduce the product cost in Avionics industry. Flight Management System (FMS), the project we are working on is long duration project where the development and verification is done parallel which spans for 2 years. There are many initiatives taken by the verification team to reduce the testing effort.

1.2.1 FEASIBILITY STUDY

A detailed study was conducted to determine if there can be any improvements that can be done to the existing process to reduce the time spend on testing. Based on the study we came up with a set of parameters that decide the feasibility of automating the testing process.

1. Is there any code that follows a specified pattern?
2. Are there any modules that are generated by the Tool?
3. Do we have any Protocol specific code which has some similarity in it?

Based on the study, we decided that testing of a Functional Area named Input/Output (IO) can be automated.

Functional Area	Approx Lines of Code	% of code following any Pattern	% of code that is tool generated	% of code that is protocol specific
Lateral Guidance	20,000	5%	0%	5%
Vertical Guidance	6997	0%	0%	5%
Performance	34795	10%	0%	5%
Crew Interface	14500	30%	25%	10%
BITE	5800	40%	0%	40%
Performance Data Base	12500	60%	0%	60%
Input / Output (IO)	59509	80%	80%	80%
Flight Plan	35120	10%	0%	10%
Database Access Manager (DBAM)	11000	10%	0%	5%

2.0 Overview of IO

Input/Output (IO) handles Input and output processing within a software partition. IO handles Software processing of I/O data between its SW access point (e.g. IMM, A653 sampling port) and the application functions (e.g. Navigation, Guidance, Performance). This includes:

- ◆ Decode/Encode

- ◆ Validation
- ◆ Source Selection
- ◆ Signal Processing (range, basic filtering, strike counting)

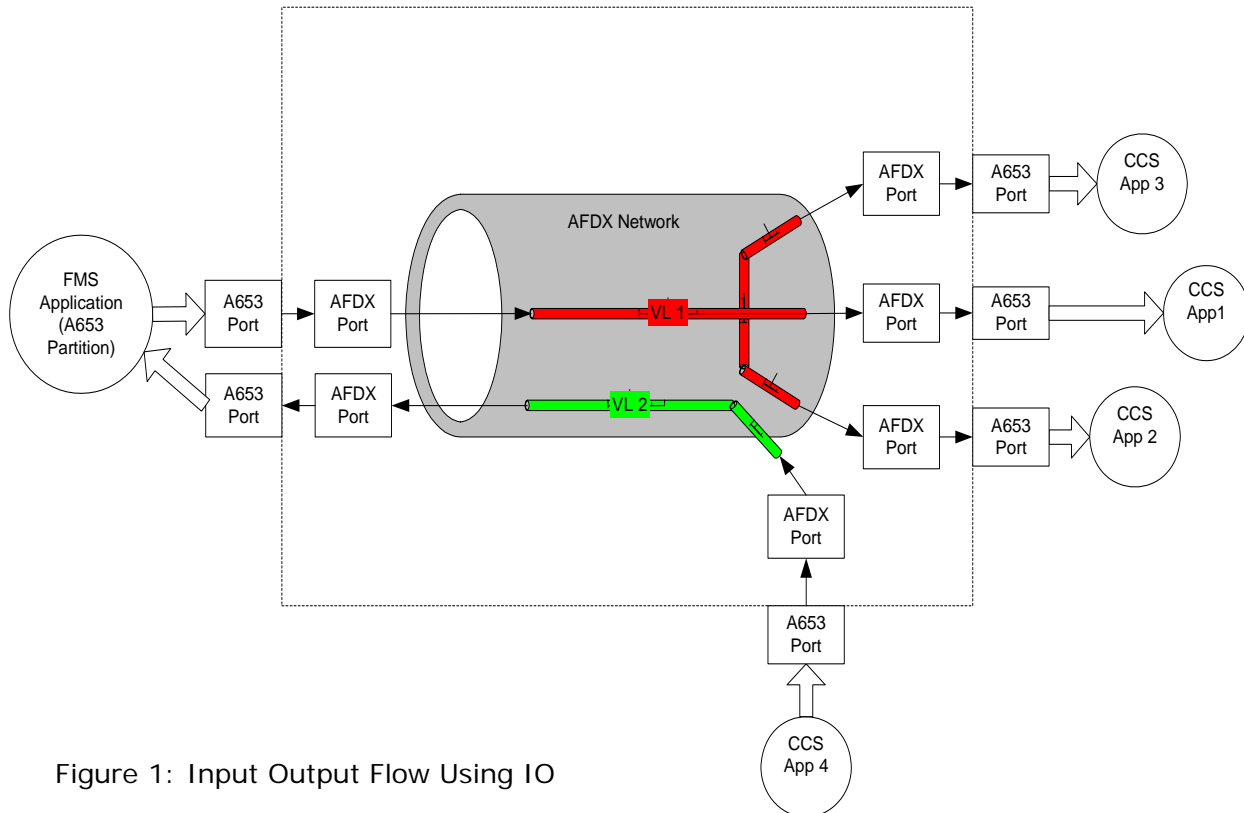


Figure 1: Input Output Flow Using IO

The Data that needs to be processed by IO is decomposed into Messages based on their type and usage. Each Message has a name associated with it and is comprised of a set of parameters. Each parameter in the message is associated with a data types and a set of permissible values based on the purpose. Broadly Messages in IO is classified into two types

- Input Processing Message
- Output Processing Message

2.1 Input Processing:

Input Processing deals with the messages given by the external devices that are used by the Flight Management System. The work of IO is to get the messages,

decode them, validate them and assign it to variables used by FMS. The flow of Data for input messages is shown in Figure 2.

Input

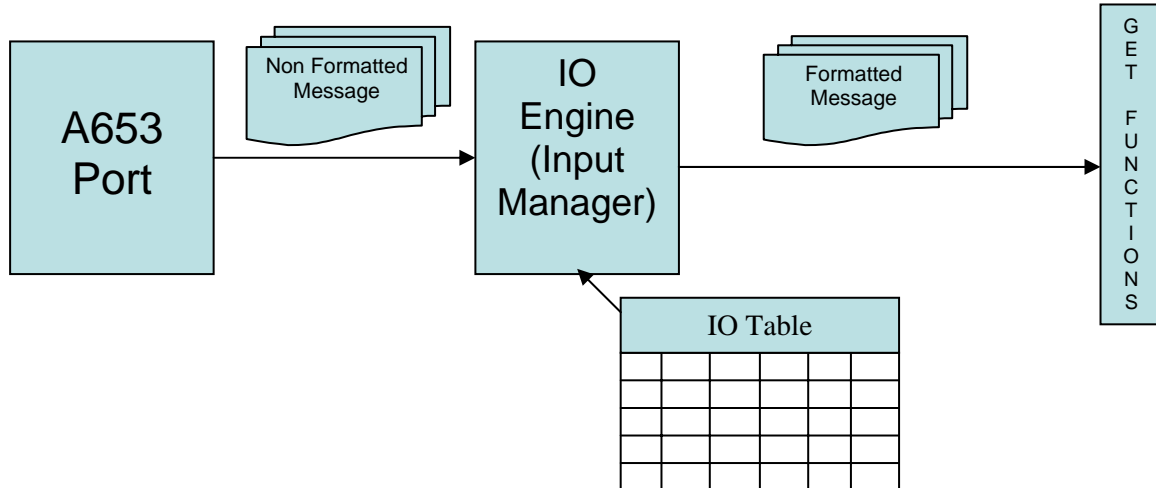


Figure 2: Processing Input Messages

2.2 Output Processing:

Data that is computed by Flight Management System and used by external devices are termed Output Messages. The work of IO is to gather the messages from FMS, validate them and Pack them into messages and send it to the external devices to use. The flow of Data for Output messages is shown in Figure 3.

Output

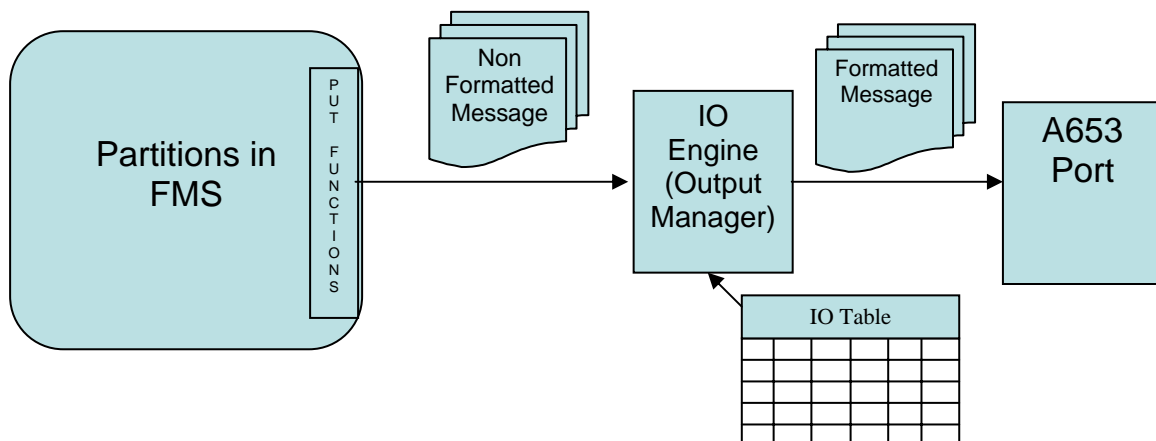


Figure 3: Processing Output Messages

2.3 Testing IO Messages:

Each message in IO needs to be verified for the following conditions.

1. Decoding/Encoding the Message as per the message model.
2. Range Check on each of the parameters in the Message.
3. Strike Counting of the parameters and the message for deciding the validity of the parameter.

Following Artifacts form the test Suite that tests the IO messages.

1. Test Scripts (Test Definition File – TDF) are developed that has the Inputs and Outputs for testing the parameters that form the part of the message.
2. Driver File that has the code for creating the ports, defining the parameters that form the part of the message and call the appropriate functions that process the messages.

2.4 Effort Spent on IO Verification:

Due these exhaustive testing that needs to be specified the size of the test scripts used to be large and lots of effort is spent on verification. Following Data is collected from the previous Aircrafts models where manual testing was performed.

Effort Spent on testing IO Manually:	15,000 Hrs
Initial Testing Effort per parameter:	5 Hours
Further Updatons of the parameter:	1 Hour

3.0 Automation using the Tool:

The proposed solution of testing the Messages was using a Tool that will generate the Test Scripts and the supporting documents. The Tool will capture the details about the message and generate the files that will test the message and each parameter in the message for all the conditions stated above.

The User Interface of the Tool is designed in Visual Basic 6.0 and the Generation of the Test Suite is done using the Dynamic Link Libraries. There are four different DLL's created to handle different message types. The Input that the user feeds is converted into a text file. This text file is then passed to one of the DLLs that convert them into Test Scripts. The User Interface decides the DLL to be called based on the type of message. The text file brings in re-usability to the software, where it

can be loaded back into the tool and modified based on the changes in the requirements.

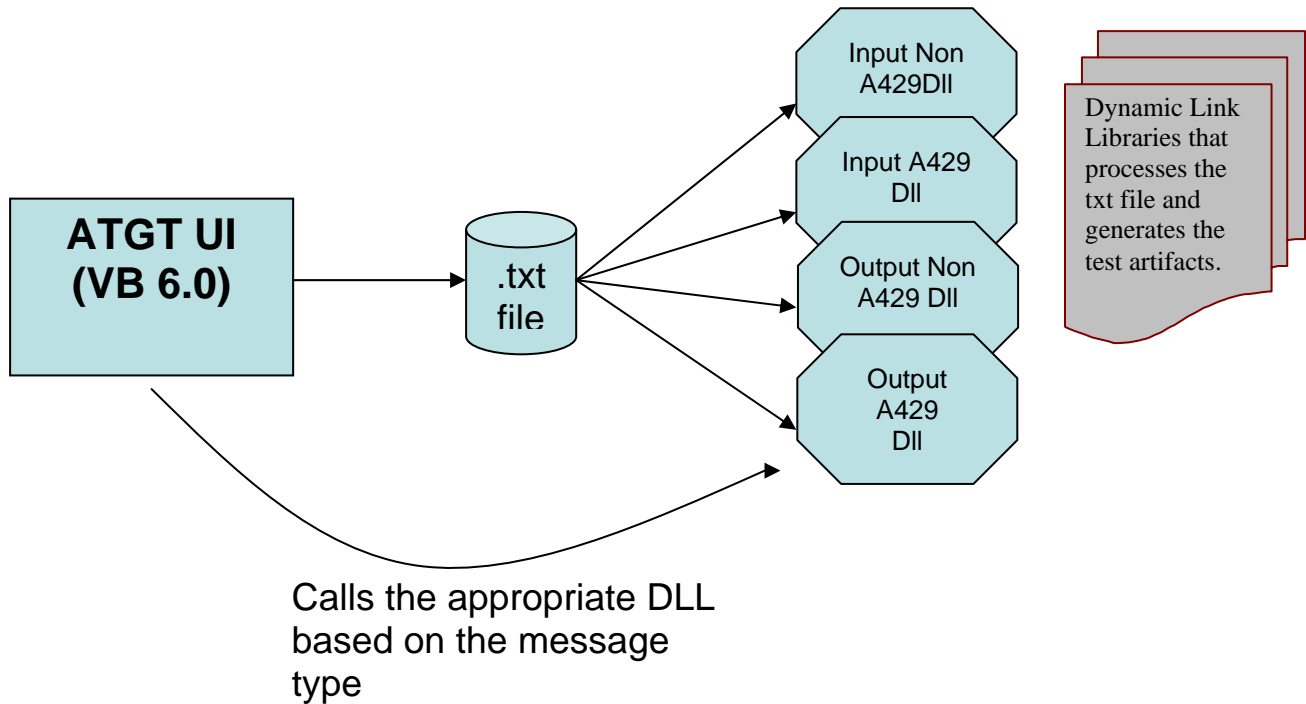


Figure 4: Tool Overview

3.1 Data types of Parameters:

The parameters of the message are associated with one of the Data types as described in the sub sections. The testing strategy is different for Each Data type and the Tools handles it based on the Input Specified by the User.

3.1.1 INTEGER:

Subtypes of Integer Data type

Sub Type	Min Value	Max Value
Int8	-128	127
Int16	-32768	32767
Int32	-2^{31}	$2^{31} - 1$
Int64	-2^{63}	$2^{63} - 1$
UInt8	0	255
UInt16	0	65535

UInt32	0	$2^{32}-1$
UInt64	0	$2^{64}-1$

3.1.2 FLOAT:

Subtypes of Float Data type

Sub Type	Min Value	Max Value
Float32	$-(2-(1/2^{22})) * (2^{127})$	$(2-(1/2^{22})) * (2^{127})$
Float64	$-(2-(1/2^{51})) * (2^{1023})$	$(2-(1/2^{51})) * (2^{1023})$

3.1.3 CODED:

Subtypes of CODED Data type

Sub Type	Min Value	Max Value
Coded8	0	255
Coded16	0	65535
Coded32	0	$2^{32}-1$

3.1.4 DISCRETES:

Subtypes of DISCRETES Data type (Boolean)

- a) One bit discrete
- b) Two bit discrete

3.1.5 OTHER DATA TYPES:

- STRINGS
- PRIVATE
- OPAQUE

3.2 PERFORMING RANGE CHECK:

Parameters of Integer, Float and Coded are associated with Min and Max values based on their usage. String/Opaque/Private parameters are associated with Length. Test Cases generated by the tool tests the parameters for their range based on equivalence class partitioning.

Here are the list of Iterations that tests the Integer/Float and Coded Parameters.

Iteration	Input	Output Value	Output Validity
I	Range Min	Range Min	TRUE
II	Below Range	Range Min	TRUE
III	Within Range	Within Range	TRUE
IV	Range Max	Range Max	TRUE
V	Above Range	Range Max	TRUE

For string/Private and Opaque parameters, Test cases are designed to test their Length.

Iteration	Input (String Length)	Output Value	Output Validity
I	0	0	TRUE
II	Length/2	Length/2	TRUE
III	Length/2	Length/2	TRUE
IV	Length	Length	TRUE
V	Length+5	Length	TRUE

3.3 STRIKE COUNTING:

Each Parameter is associated with the acceptable level of tolerance for the Bad samples. If there are instances of consecutive bad samples that exceeds the tolerance level the output of the message should be ignored as the trust on the source from where the message comes is lost. And in the case when it generates acceptable number of good samples again, it gains the trust and the message can be used for processing. There are properties associated with each parameter which decides the Tolerance of bad and good samples. These properties vary from one parameter to another based on their importance.

Few of the important properties associated with the parameters are:

1. Fail Count - Number of consecutive Bad samples that will make the parameter untrustable.
2. Pass Count – This decides the number of consecutive good samples that will bring back the trust.
3. Strike Counter State – This parameter holds the state of the current message if it is good or bad.

The Automated Tool generates the Test case that tests all the conditions given above.

3.4 Design of Driver:

This automated tool generates the C file that acts as a driver to test the messages. This driver when executed will make the test conditions ready and call appropriate functions that help to tests the output.

3.4.1 Driver Design for INPUT Messages:

This block diagram explains the driver design for INPUT messages.

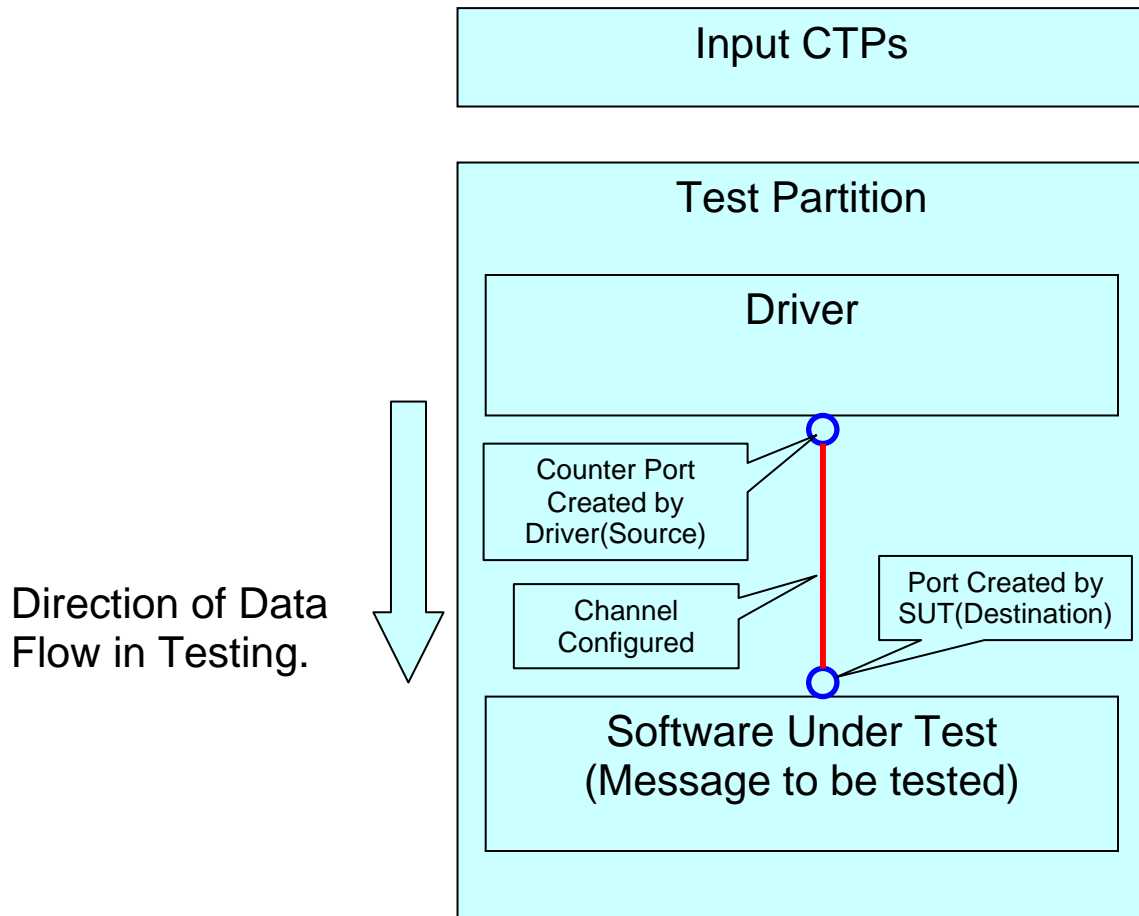


Figure 5: Driver Design for Input Messages

Each message is associated with a Port through which the data flows. The Direction of the port depends on the Type of message. For an Input Message, the port present in the code (To be tested) is a Destination port, so to test the message, we need to create a Source port, pass input through the port and check if the data is received correctly. Ports are associated with unique names through which they are identified. Communication between the ports is established through channels.

3.4.2 Driver Design for OUTPUT Messages:

This block diagram explains the driver design for INPUT messages.

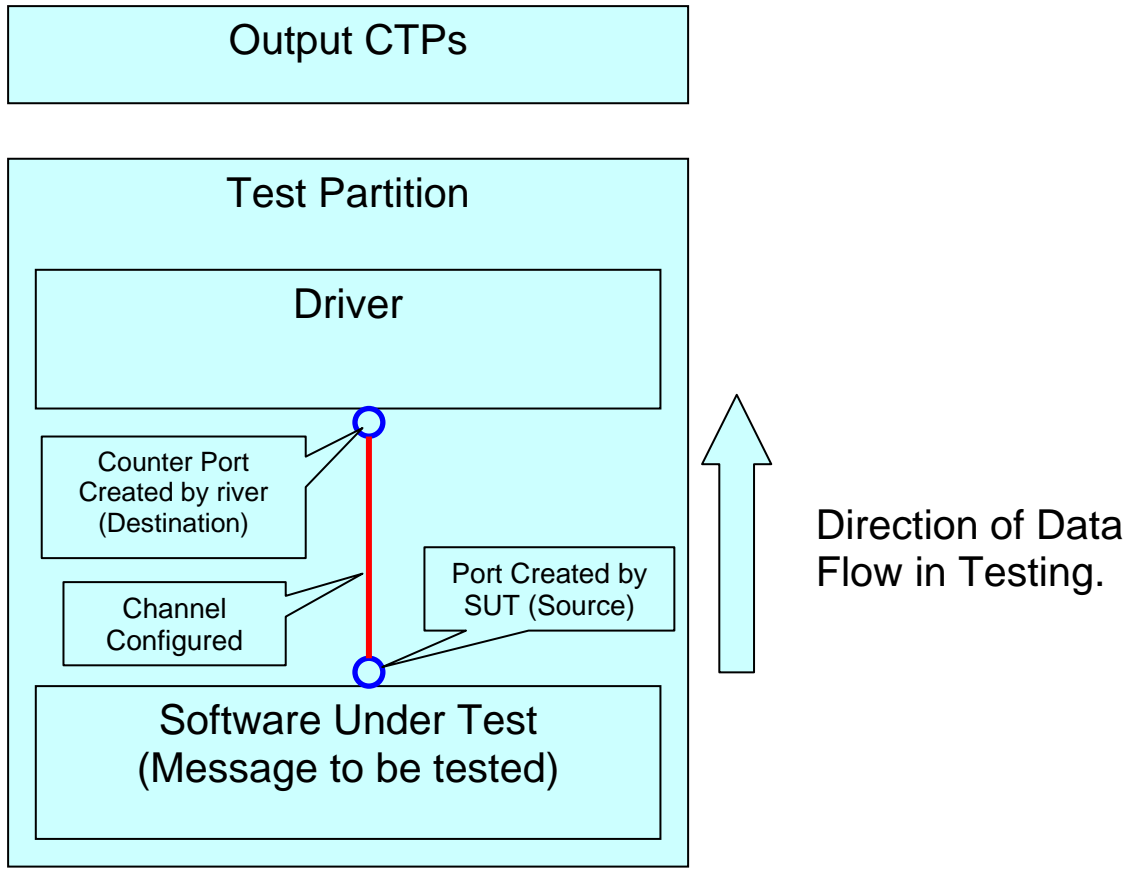


Figure 6: Driver Design for Output Messages

For testing Output Messages, The driver needs to create a Destination port, for receiving the messages that is sent by the Code (To be tested).

3.5 Sample Message Format from the Requirement Document.

Name: MSG01_ADRF_AFDX			
Processing Rate: 50	Frames: even		
Golden FS (Label): FMF_DS1_FS_1	Partitions: FMF	Is A429: Y	Periodic: Y
Config Inhibit Mask: 64			
Config Inhibit Expected: 64			
Config Inhibit Parameter: 40			
Condit Inhibit Event: N/A			
Source Selection: ADRF_Selection		Post Processing: N/A	
Instance: Left	A653 Port: S_FMF_L_AIR DATA_1		
Instance: Right	A653 Port: S_FMF_R_AIR DATA_1		
Instance: Center	A653 Port: S_FMF_C_AIR DATA_1		

Functional Status	ICD Name	Pass Count	Sample Count	Fail Count	Golden
FS8	FMF_DS1_FS_8	3	3	3	F

ICD Parameter Name: ADRF_L_ALTRATE_VOTED					
Data Dictionary Name: IO_ADRF_L_Altitude_Rate_Voted					
Message Name: MSG01_ADRF_AFDX					
API Data Type: FLOAT		Positive Sense: North		Functional Status: FMF_DS1_FS_8	
Units: ft/min				Resolution: 11	
Min Range: 0			Max Range: 200		

(NOTE: Few Parameters corresponding to the document given are not explained as they are not relevant to the Tool)

3.6 Tool Screen Snapshots:

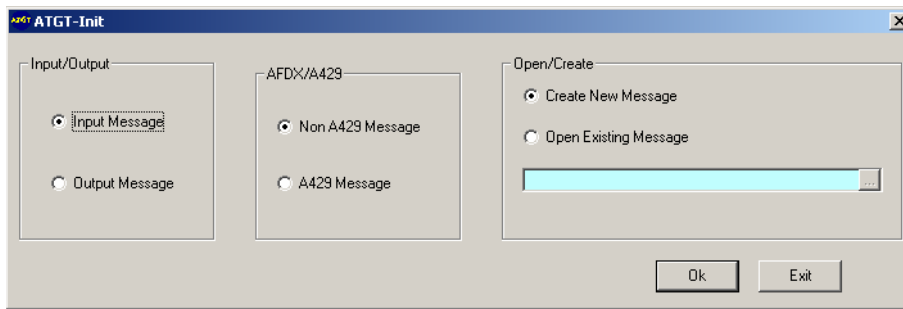


Figure 7: Initialization Screen of the TOOL.

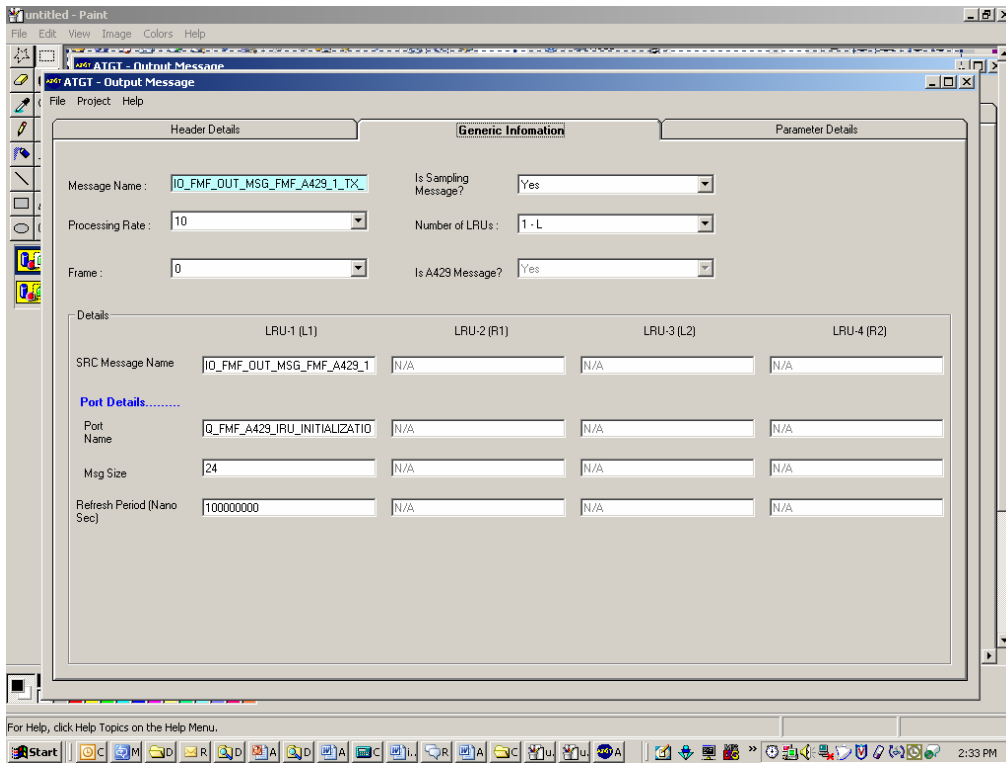


Figure 8: Inputting Message Details using the Tool.

3.7 Effort Savings Using the Tool:

The Tool is developed and tested and used by Boeing and Airbus verification team to generate Test Suite for the IO Messages and has resulted in 10,000 Hours of Effort savings.

Effort for Manual Testing: 15,000 Hours
 Effort using the Tool: 3,400 Hours
 Tool Development Effort: 1,600 Hours

Effort saved on date: 10,000 Hours (66% Effort savings)

This Tool is generic enough to be used by other programs that will result in more effort savings.

The concept of Automation is presented as a Best Practice at Domain Level and the modified version of the Tool is proposed for domains. This concept can be implemented across domains, and companies which will result in huge effort savings and also improve the quality of Test. The Goal of Defect Free product can be achieved using Automation tools.

APENDIX:

AVIONICS: means aviation electronics. In essence it comprises all electronic systems designed for use on an aircraft. At a basic level this comprises communications, navigation and the display and management of multiple systems.

FMS : Flight Management System or FMS is a computerized avionics component found on most commercial and business aircraft to assist pilots in navigation, flight planning, and aircraft control functions.

Line-replaceable unit (LRU) is a black box of electronics, such as a radio or other auxiliary equipment for a complex engineered system like an airplane or ship. LRUs speed up repair, because they can be replaced quickly, restoring the big system to service.

Aeronautical Radio, Incorporated (ARINC), established in 1929, is the leading provider of transport communications and systems engineering solutions for five major industries: (aviation, airports, defense, government and transportation).

ARINC 653 (A653) is a standard for partitioning of computer resources in the time and space domains. The standard also specifies APIs for abstraction of the application from the underlying hardware and software.

Avionics Full-Duplex Switched Ethernet (AFDX) is Part 7 of the ARINC 664 Specification which defines how Commercial Off-the-Shelf (COTS) networking technology will be used for future generation Aircraft Data Networks (ADN). AFDX defines a low-level network and protocol to communicate between avionics (referred to as End-System) devices in aircraft. It is based on Ethernet, and like all full-duplex networks uses dedicated outgoing and incoming channels to allow full-speed transmission in both directions at the same time. AFDX extends standard Ethernet to provide high data integrity and deterministic timing. It specifies interoperable functional elements at the following OSI Reference Model layers:

IO TABLE

The Details about the parameters and statically defined in the form of tables that is processed by the engine to convert them into messages.

ABBREVIATIONS:

FMS – FLIGHT MANAGEMENT SYSTEM
SW – SOFTWARE
IMM - INTER MODULAR MEMORY
INT - INTEGER

UINT - UNSIGNED INTEGER
MAX - MAXIMUM
MIN - MINIMUM