Automation of Tester Diagnostics

Shailendra S, Vijaya Prakash A M, Nishanth Ravindranth

Abstract: Tester needs to test software in ECU or implement changes in it and then test software depending on the client's requirement. To test many real time scenarios, he makes use of ETAS Labcar which helps us to read and write different variables and validate test cases. He needs to set up the different tools and software to flash and validate the software which takes more time. For this we make use of ECU-Test tool which helps us to automate the process and we can run everything in one place. It has workspace which is specific to specific client project. Here we introduce new standard workspace which helps to easily switch between projects and helps the tester to save a lot of time. We also introduce the Basic component regression package which helps us to identify any errors found in Basic component level which might be hard to find otherwise.

Keywords: AUTOSAR, UDS, ETAS Labcar, INCA, ECU-Test, BC Regression.

I. INTRODUCTION

Before the introduction of Electronics in Automotive Industry, automobiles were seen as mechanical machines. Accuracy was the main limitation in the mechanical systems and to overcome this Electronic Control Unit (ECU) was introduced. In automobiles, ECU is an embedded system which controls one or more electrical system or subsystem. There are many varieties of ECU which is used to control different section of automobiles like brake, powertrain, suspension, engine etc. More than 70 ECU are present in modern vehicles. For communications between these ECUs, previously LIN **[5]** (Local Interconnect Network) which is a serial network protocol was used between components in vehicles. But due to wire complexity and other disadvantages, we now use CAN **[4]** (Controller Area Network) which is more robust vehicle bus for communication.The OEM's (Original Equipment Manufacturer) will provide requirements, which may be implementation of new software or modification in the given software. These software or program versions are stored in software configuration management (SCM). It maintains the current state of software, while helping the developers to work on fixes or new versions for features. SCM can tell what was changed and who changed it, whenever something went wrong. Tester will extract the required program versions and do changes according to the need of the OEM. Later this program version is built to produce a2l, hex and elf files. Any error in build must be resolved.

Revised Manuscript Received on July 15, 2020.
* Correspondence Author
Shailendra S*, Student, VLSI and Embedded Systems, Bangalore Institute of Technology, Bengaluru, Karnataka, India.
Dr. VijayaPrakash A M, Professor, VLSI and Embedded Systems, Bangalore Institute of Technology, Bengaluru, Karnataka, India.
Nishanth Ravindranth, Specialist (EEP, BOSCH), VLSI and Embedded Systems, Bangalore Institute of Technology, Bengaluru, Karnataka, India.

The a2l & elf contain the variables and other information. Hex file is used to flash the ECU. Programming to ECU require a hex file which has all the values, variable, functions, mappings etc.The hex file is a hexadecimal source file used by programmable logic devices. The hex file is generated after building a program version. A program version contains all the files such as C, XML and ARXML which contain codes, values, functions, mapping etc.Flashing of hex file to ECU is done usually through Universal Debug Engine (UDE) or through Integrated Calibration and Application Tool (INCA). Once the software is flashed, a tester can check if it is working properly as expected by checking the results through frames by using CANanalyzer tool. We here automate the above process using a tool called ECU-Test. ECU-Test is a software tool for test and validation of embedded systems. In ECU-Test we will have workspace where we can write different test cases and validate. This workspace is specific for specific client project.

II. LITERATURE SURVEY

- In automobiles, the Electronic control unit (ECU) is considered as one of the main units which takes input from sensors and other signals and provide the signal for the actuators to work. **[1]** Based on electro-hydraulic servo control, ECU of blend brake system for heavy vehicle is designed. A blend brake consists of three basic components: Hydraulic decelerator, disc brake system and electronic control unit. Here, disc brake should not be in action when speed is higher than the setting value but be involved in at emergent state. In emergent state, the signals from velocity transducer be collected by ECU and as a result the logical control between hydraulic decelerator and disc brake is competed, so the park brake is achieved.
- In this system **[2]**, they have used Electronic Control Unit (ECU), adaptive cruise control (ACC) and engine management system (EMS)**.** Adaptive Cruise Control measures the distance of the user's vehicle from the front vehicle, by using IR sensor. The corresponding distance data is given to the Engine Control Unit. Depending on the distance gap and the motion of the front vehicle the ECU takes the decision. The engine control system receives input signals from the different sensors that measure the condition of the engine. With the help of these signals, the controller generates output signals to the actuators which determine the engine calibration.
- In heavy diesel engine, Electronic unit pump (EUP) fuel injection system has important application. In order to meet the emission regulations and economic requirements, developing an ECU **[3]** of EUP diesel engine is much required. EUP which has high speed fast-response solenoid valve is a major part which controls the fuel injection timing and injection quantity.

ECU maintain different modules like power, digital circuit etc.
where it takes the input and provides the output where the fuel ejection is checked. The fuel consumption changes gradually between 225g/kW·h to 235g/kW·h. It shows good economy of the EUP engine.

- Electronic devices for communication and computer engineering have implemented X-by-wire scheme for braking and steering control in automobiles. **[4]** This paper proposes an event trigger design for vehicle control by using the protocol of Controller Area Network (CAN). They proposed design for CAN communication mode and interactive GUI-based utility program which helps user to easily develop the required function call to cope with design specifications.
- Nowadays in modern automobiles, Controller Area Network (CAN) is more and more widely adopted. This automobile network, a system that is able to reduce the number of wiring harnesses and realize the sharing of data is more efficient. This paper adopts CANalyzer **[5]** to avoid making serious mistakes in hardware connection, to simulate the study of the practicability and reliability of this system and to cut down experiment costs. It is used to transmit interactive frame message, display data of certain message, track bus data, generate statistical list, record data and check bus malfunctions etc.
- Nowadays Controller Area Network (CAN) is used to communicate between the Electronic Control Units (ECU). It is reliable and robust. But the diagnostics of specific ECU is a complex task, when the number of on-board ECU is increased. The diagnosis of the electronic devices and electrical circuits is done with a standardized communication protocol, Unified Diagnostic Services (UDS) **[6]**. If such UDS application is present on every ECU in the car, the I/O testing can be done via an external tester easily. Under UDS 26 services are supported including “Tester Present”, “Diagnostic Session Control”, "Write data by address", “Read Data by Identifier”, "Input/Output control" etc.
- To tune the parameters to meet the required industry standards or to either achieve/evaluate the required functionality, to understand the behaviour of vehicle dynamics in advance, the only way is to perform simulation. The major challenge is even if information made available, proper combination of simulation tools with Real-time capability. The closed loop simulation of complex dynamic systems with the real time capability can be brought by the Real time simulation using Hardware-in-Loop (HiL) technology. The usage of AVL-CRUISE with CarMaker working in Real-time environment of ETAS Labcar **[7]** for the analysis of complicated vehicle systems is being presented in this paper. This will help to enhances the real time feature in road to be tested in LAB.
- Tester needs to check if he is obtaining the required frames the ECU. As discussed in **[8],** the developed application aims to ease the testing process performed over the diagnostic services on CAN. Diagnostic service requests can be generated with the help of automated tests and one can determine the similarities between customer requirements and the current implementation, through the feedback from the ECU.
- With the increasing features in automobiles, there is increased number of parameters to keep check and hence is the requirement of diagnostic system. This system must contain protocol for connecting different tools used by designer or testers in order to check ECU information. For this reason, we use protocol such as CAN, KWP2000 and UDS. This paper **[9]** provides the overview of the protocols at different layers of network.
- Source code need to be tested as it decreases the development cost. This method depends on Hardware in loop simulator. As testing in this manner takes long time and needs to be tested multiple times, we go for software in loop simulation **[10]** in this paper. Here simulation of software component will be working in virtual prototype in automobile and corresponding behaviour is recorded. Here author has used automatic emergency braking system as case study and showed the efficiency in software in loop.

III. INSIGHTS ON AUTOSAR, UDS, POWERTRAIN

3.1 AUTOSAR

Every new era of autos has new ideas being included; the frameworks are getting significantly more complicated thus check is essential at the framework level as well as at the part level. The product of all ECUs has a million of lines-of-code. The main objective of AUTOSAR is to establish a common standard among the software suppliers, manufacturers and tool developers, keeping the competition so that the final outcome of business is not altered in the process. AUTOSAR's layered architecture is shown Figure 3.1.

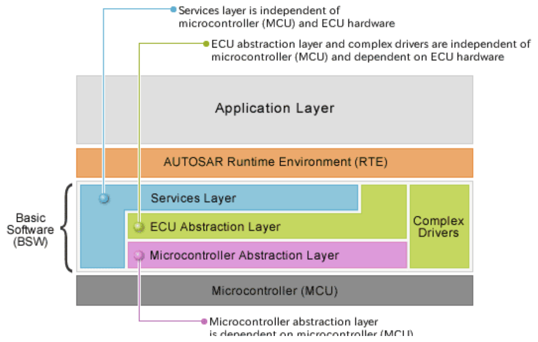

Services layer is independent of microcontroller (MCU) and ECU hardware
ECU abstraction layer and complex drivers are independent of microcontroller (MCU) and dependent on ECU hardware
Application Layer
AUTOSAR Runtime Environment (RTE)
Basic Software (BSW)
Services Layer
ECU Abstraction Layer
Microcontroller Abstraction Layer
Complex Drivers
Microcontroller (MCU)
Microcontroller abstraction layer

Figure 3.1 - AUTOSAR Layered Architecture.

RTE (Runtime Environment) is a middle layer which manages the intra and inter ECU communication between application layer components as well as between the BSW and the application layer. BSW (Basic Software) is defined as standardized software module providing different services required to run the functional part of the upper software layer. This layer is divided into three sub layers which are, Services layer, ECU abstraction layer, and the Microcontroller Abstraction Layer (MCAL). Services Layer is highest BSW layer that provide Operating system, Vehicle communication and management, Memory services (NVRAM management), Diagnostic service (UDS) and ECU state management. It provides basic services and software modules for application.

3.2 Unified Diagnostic Service (UDS)

Standard diagnostic protocol is needed because the OEMs and suppliers had to deal with compatibility issues between different diagnostic protocols **[9]** like ISO 15765, KWP 2000, and diagnostics over K-Line. UDS **[8]** is an automotive protocol which lets the diagnostic systems communicate with the ECUs to diagnose faults and reprogram the ECUs as and when required. As it combines all the standards like ISO 15765, KWP 2000 and others, it is called unified. Its features include, Data transmission, fault diagnostics, remote routine activation, upload or download data.

3.3 Powertrain

In a motor vehicle, the main components that generate power and deliver that power to the road surface, water, or air is the **powertrain** or **powerplant**. This includes the drive shafts, transmission **[2]**, engine, differentials, and the final drive (continuous track (like in caterpillar tanks or military tractors), drive wheels, propeller, etc.).
BOSCH Group was founded in 1886 as a workshop for precision Mechanics and Electrical engineering by Robert BOSCH Group in Germany. The BOSCH Group today is the largest automotive technology supplier in the world with a global group turnover of 78 billion. The department **RBEI/EEP** was made in February 2011 to majorly support the Customer projects from DGS (Diesel Gasoline Systems). EEP supports DGS in developing control units for diesel and gasoline engines, alternative power drives and also works for powertrain solution, engine ECU and emission control standards. This includes project management, requirements analysis, planning and integration which are important task on the top half of the V-model. Powertrain electrification is one of the most important strategies of every OEM in achieving the long-term goal of carbon neutral mobility.

IV. METHODOLOGY

4.1 Workflow

The workflow involves getting requirements from OEM, validation of those requirements, coding and implementation of requirements in program versions, building of program versions, flashing the ECU, testing UDS services, checking the functionality and analyzing its working in real time environment with the help of different tools available.
Basically, build is the process of creating the application program for a software release, by taking all the relevant source code files and compiling them and then creating a build artifact, such as binaries or executable program, etc. Tester need to note all the possible outcomes. He needs to first analyse different test cases beforehand and provide what are the possible expected outcome which should satisfy the UDS standards as well as requirement of OEM. He should check if tests **[1,3]** pass or fail. If a test fails, then he needs to confirm if test has failed as he expected. If not then he needs to analyse the situation and solve the bugs. Even if test has passed and it is not providing the expected value then he needs to debug the cause and modify the software again and do the whole process again. After obtaining the hex file from building the program version, flashing is usually done through UDE or INCA tool.

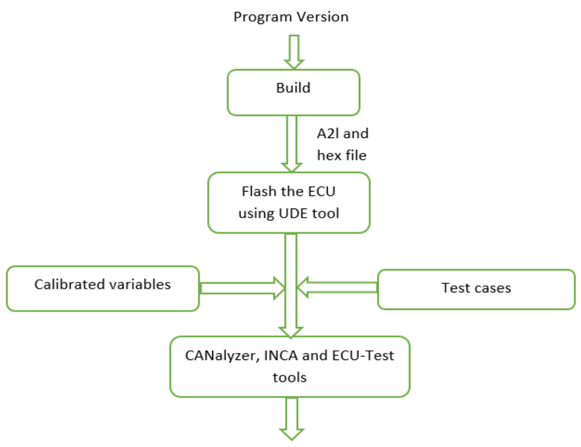
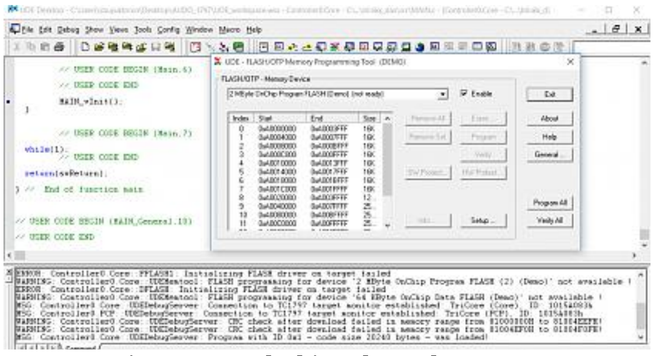


Figure 4.1 - Workflow in Powertrain department.

After flashing, communication with ECU is done through CANalyzer tool. Here initially, we need to set the configuration for CAN, and then we can send the requests to ECU as frames. These frames must follow UDS standards **[6]**. ECU will respond with corresponding frames which also follows UDS standard. For example, if we requested for extended diagnostic session that is the frame 10 03, then ECU should respond with frame 50 03. Later Tester needs to check different test cases and confirm if ECU is responding as expected.
To check real time scenarios, we use INCA **[7]** tool which helps us to calibrate different variables and set these variable values to that which matches real time scenario. We can calibrate these variables without affecting rest of the program files. Later we can measure these variables to confirm if it has the desired values and then proceed with testing different test cases.
All these steps can be integrated and the process can be automated **[8]** using ECU-Test tool. Here we can interface with different tools like INCA, CANalyser and other tools. Here tester can write different test cases and can check all the results easily. Since we have interfaced with CANalyzer and INCA, we can easily calibrate different variables and check different test conditions and send frames to check the ECU's working all in one place **[10]**. We need to set up a workspace in beginning. This is specific to specific project.


File Edit Debug Show Views Tools Config Window Macro Help
UDE - FLASH/OTP Memory Programming Tool (DEMO)
FLASH/OTP - Memory Device
Enable
Exit
About
Help
General ...
Setup ...
Program All
Verify All

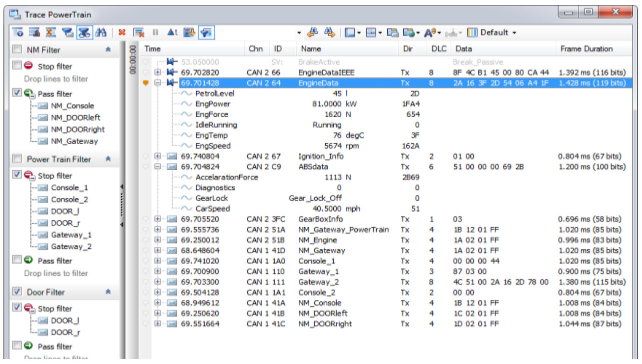
```
    // USER CODE BEGIN (Main,6)
    // USER CODE END
    MAIN_vInit();
  }

    // USER CODE BEGIN (Main,7)
  while(1);
    // USER CODE END
  return(swReturn);
} // End of function main

// USER CODE BEGIN (MAIN_General,10)
// USER CODE END
```

2 MByte OnChip Program FLASH (Demo) (not ready)

Index	Start	End	Size
0	0xA0000000	0xA0003FFF	16K
1	0xA0004000	0xA0007FFF	16K
2	0xA0008000	0xA000BFFF	16K
3	0xA000C000	0xA000FFFF	16K
4	0xA0010000	0xA0013FFF	16K
5	0xA0014000	0xA0017FFF	16K
6	0xA0018000	0xA001BFFF	16K
7	0xA001C000	0xA001FFFF	16K
8	0xA0020000	0xA003FFFF	12...
9	0xA0040000	0xA007FFFF	25...
10	0xA0080000	0xA00BFFFF	25...
11	0xA00C0000	0xA00FFFFF	25...

```
ERROR: Controller0.Core::PFLASH1: Initializing FLASH driver on target failed
WARNING: Controller0.Core::UDEmemtool: FLASH programming for device '2 MByte OnChip Program FLASH (2) (Demo)' not available !
ERROR: Controller0.Core::DFLASH: Initializing FLASH driver on target failed
WARNING: Controller0.Core::UDEmemtool: FLASH programming for device '64 KByte OnChip Data FLASH (Demo)' not available !
MSG: Controller0.Core::UDEDebugServer: Connection to TC1797 target monitor established: TriCore (Core), ID: 1015A083h
MSG: Controller0.PCP::UDEDebugServer: Connection to TC1797 target monitor established: TriCore (PCP), ID: 1015A083h
WARNING: Controller0.Core::UDEDebugServer: CRC check after download failed in memory range from 81000000H to 81004EEFH!
WARNING: Controller0.Core::UDEDebugServer: CRC check after download failed in memory range from 81004EF0H to 81004F0FH!
MSG: Controller0.Core::UDEDebugServer: Program with ID 0x1 - code size 20240 bytes - was loaded!
```

Figure 4.2 - Flashing through UDE



```
Time            Chn   ID    Name                     Dir  DLC  Data                       Frame Duration
   53.050000          SV:   BrakeActive                        Break_Passive
⊞  69.702820    CAN 2 66    EngineDataIEEE           Tx   8    8F 4C B1 45 00 80 CA 44    1.392 ms (116 bits)
⊟  69.701428    CAN 2 64    EngineData               Tx   8    2A 16 3F 2D 54 06 A4 1F    1.428 ms (119 bits)
     PetrolLevel                      45 l           2D
     EngPower                    81.0000 kW          1FA4
     EngForce                       1620 N           654
     IdleRunning                 Running             0
     EngTemp                          76 degC        3F
     EngSpeed                       5674 rpm         162A
⊞  69.740804    CAN 2 67    Ignition_Info            Tx   2    01 00                      0.804 ms (67 bits)
⊟  69.704824    CAN 2 C9    ABSdata                  Tx   6    51 00 00 00 69 2B          1.200 ms (100 bits)
     AccelarationForce              1113 N           2B69
     Diagnostics                       0             0
     GearLock               Gear_Lock_Off            0
     CarSpeed                    40.5000 mph         51
⊞  69.705520    CAN 2 3FC   GearBoxInfo              Tx   1    03                         0.696 ms (58 bits)
⊞  69.555736    CAN 2 51A   NM_Gateway_PowerTrain    Tx   4    1B 12 01 FF                1.020 ms (85 bits)
⊞  69.250012    CAN 2 51B   NM_Engine                Tx   4    1A 02 01 FF                0.996 ms (83 bits)
⊞  68.648604    CAN 1 41D   NM_Gateway               Tx   4    1A 02 01 FF                1.020 ms (85 bits)
⊞  69.741020    CAN 1 1A0   Console_1                Tx   4    00 00 00 44                1.020 ms (85 bits)
⊞  69.700900    CAN 1 110   Gateway_1                Tx   3    87 03 00                   0.900 ms (75 bits)
⊞  69.703300    CAN 1 111   Gateway_2                Tx   8    4C 51 00 2A 16 2D 78 00    1.380 ms (115 bits)
⊞  69.504128    CAN 1 1A1   Console_2                Tx   2    00 00                      0.804 ms (67 bits)
⊞  68.949612    CAN 1 41A   NM_Console               Tx   4    1B 12 01 FF                1.008 ms (84 bits)
⊞  69.250620    CAN 1 41B   NM_DOORleft              Tx   4    1C 02 01 FF                1.008 ms (84 bits)
⊞  69.551664    CAN 1 41C   NM_DOORright             Tx   4    1D 02 01 FF                1.044 ms (87 bits)
```

Figure 4.3 - CAN frames in CANalyzer

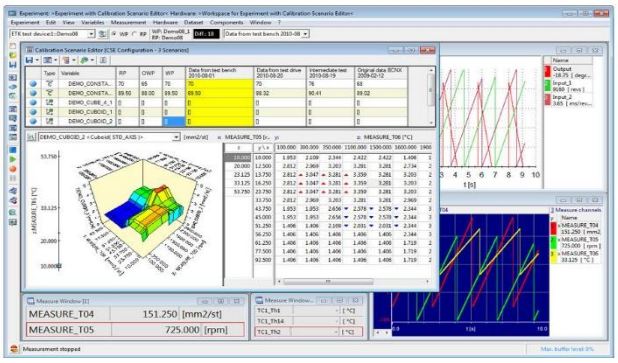


Figure 4.4 - Experiments in INCA tool

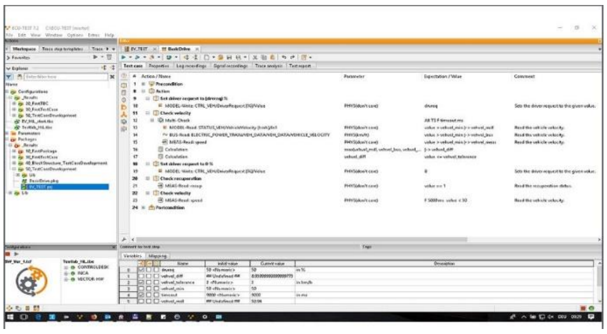


Figure 4.5 - Workspace of ECU-Test tool

4.2 Current Issues

- **Specific workspace for specific project:** For Each Projects we will have different workspace and they are specific to that project. Once we have set a workspace and interfaced with different tool properly, it requires some time to set new workspace for another project and to interface with different tool.
- **Configuration of INCA database:** Database is specific for specific project which contain the way of communication in transport layer. This working database needs to be given as template in tbc file and we need to provide correct container structure of how it is loaded. This take some time to set new database and provide the same in ECU-Test tool.
- **Setting controller for UDE:** We should provide the right controller in UDE for specific project. Any mismatch in controller can either not connect or damage the ECU.
- **Basic Component Regression:** The changes made in the functional component which compromises of all the codes written, can cause some changes in other functional component which might have dependencies and this leads to regression which needs to be checked at Basic component level which compromises of different functional components.

4.3 Methodology

By creating a standard workspace, we can have all workspace of different project in one workspace. Once we set workspace for one project, it will be easy to switch to another project. Thus, we reduce the tester's time to complete his work as well as use Labcar efficiently.

Creating Standard workspace:

- We first analyzed the workspace of one project and understood how all the things are related.
- We later compared all the packages of one client project with another project and found similaritie. All the packages which are similar to both the client projects are made to put under a common container and only the project specific packages are made to put under separate project container.
- We made a standard structure on how to arrange all the packages pertaining to different container. All these packages are arranged in project (.prj) of specific client project.
- We created a project where we can select the client project which we require and deselect the rest and when we run the project, it only runs the selected client project. All the tbc, tcf and flashing process related to projects are also added here.

Creating Standard database of INCA:

- We took a2l and hex of all the required project and found all the different ways of communication.
- We created a structure on how to place all the way of communication in one database, so that one can distinguish it in ECU-Test tool. This communication also depends on what type of ETAS device is connected. So, we added structure of different ETAS device.
- We created different tbc file for each different way of communication and for each different ETAS device to be connected. This way the user need not update the tbc all the time when he switches the project.
- All the tbc pertaining to different project and different ETAS are already created and user can easily select the one he needs. This reduces the time to set up database, complexity of switching and increase effective use of tools.

Presetting the Controller for UDE:

- For all the different tcf created, we added all the necessary controller required by different ECU and put them as global constant in tcf. This way the controller won't change for specific project and user need not worry about selecting wrong controller.
- We created a package where we fetch hex file from tcf and provide to jobs of UDE to flash to ECU. This will automatically flash and provide the result saying if flash was successful or not. This package is added to standard project which we created earlier.

Modified Standard Workspace:

- Before we user has to fill whether he needs to flash before starting the project. Now, we altered so that user gets a pop-up asking if he needs to flash or if it's done already. If we select 'no', then it won't open UDE and continue with validations. If we have selected 'yes', the user gets another pop-up whether he wants to provide hex file manually or if he wants to package to fetch automatically from tcf file. Once it gets the hex file, it flashes the hex file to ECU and responds with success or failure.
- We created a test configuration where it gives pop-up window where it asks user to select the project he needs. Once he selects the client project, it will automatically update the required tcf and tbc file and start with the validations.
- Once we updated the INCA database and updated all the tbc, we provided one more pop-up after selecting the client project to select which ETAS device is connected. Upon selecting the ETAS, it will automatically connect with INCA with the required way of communication and can easily move with validations.

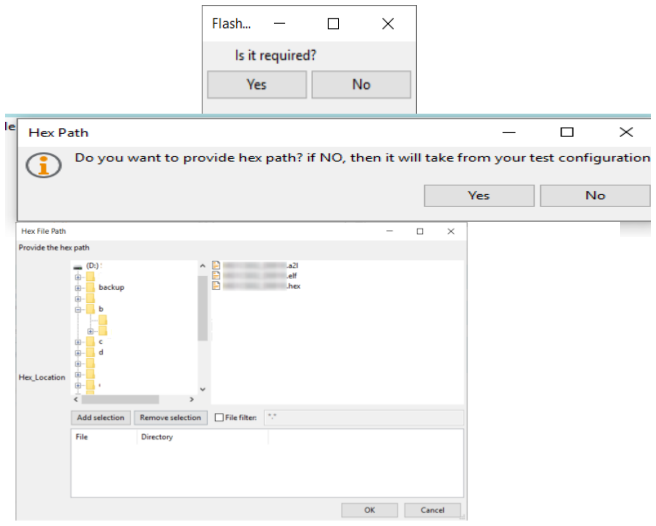

Flash...
Is it required?
Hex Path
Do you want to provide hex path? if NO, then it will take from your test configuration
Hex File Path
Provide the hex path

Figure 4.6 - Modification in Flashing process

Basic Component Regression:

Regression is measure of impact of a function over the other function. In basic component regression packages, we check how the dependency of one over the other effect. All the codes written will be put under functional component and most test cases are written to validate the process under functional component level. If there is a dependency of one over the other, the changes made in one, which might have caused error might go unnoticed. To prevent such scenarios, we have written different test cases which checks the faults under basic component level.

Some of the test cases include:

- CAN ID:
  Here we check if transmission and reception of frame, to and from ECU are in expected CAN ID's or not.
  We check
    - Physical Address
    - Functional Address
    - OBD

  For a particular project with defined address for communication. We first check if we are getting response from the physical address mentioned for communication. Here we send Tx frame and except the Rx frame in expected CAN ID. We similarly send Tx frame in functional address and OBD and check if we get response in expected CAN ID.
- Request user for Supported Services:
  Here we check if selected service by the user is supported and it provide positive response. If user does not provide the selection of services in time then by default it selects all service are supported and continue the validations. And also, the unsupported service should provide an NRC. This is required to check if the user selected services are supported or not.
  For example, if a user has requested for write data by identifier and write access is not supported. At first user will request to enter session (10 03 – extended session) and ECU will respond for session entry (50 03). Now if user tries to write some data (2E XX XX 12 34 (here, XX XX is data identifier)), the ECU will respond with NRC (7F 2E YY (here YY is corresponding negative response))
- NRC Priority:
  Negative Response Code (NRC) is generated when user sends a frame of wrong format or frame containing unexpected data. UDS standards provide a set of NRCs for particular wrong requests. There is also specific NRC set for particular services. When user provide multiple mistakes for a single frame. The ECU will respond with only one NRC. And this NRC has higher priority over the other. For example, if you have provided wrong minimum length and you have entered wrong session, ECU will respond with minimum length error NRC over wrong session NRC.
- Cancel Routine:
  When we start a routine, it ends either after finishing or by providing stop routine or by cancelling it. Routines run in extended session and does not support in default session. When the routine is running, if we request for default session, routine will cancel. But if we have made mistake in code and if routine isn't cancelled then it would cause problem for next validation. Here we check if cancellation of routine is happening or if the routine is still running. We first start a routine which has long time for completion and we enter default session. We now request for routine result and expect a NRC of request sequence error which shows that routine is cancelled.
- Activation condition for each service:
  Some services will have certain pre-condition to be fulfilled before sending a frame to get positive response. These conditions are activation conditions. There are general activation conditions and specific condition for certain routine or other services. We need these checks so as to avoid some services being passed even if the conditions are not met. This violates the requirements.
- Supported session for each service:
  Some services are supported only in certain sessions and they should provide NRC when they are requested in unsupported session.

This is to check in case we have done mistake while coding and the ECU responds even in unsupported service. To identify such errors, we request each servicer in all the sessions and check the corresponding responses. All the request sent in supported session must provide positive response and rest should provide NRC.

V. RESULTS AND DISCUSSIONS

The workspace which are specific to specific client is as shown. Here Tester has to manually interface with UDE in order to flash and later initialize INCA and set up the required database for the specific client project and later connect CANalyzer to send frames and validate the test cases. The process is tiresome and takes long time to get the required result. If the tester has found some errors then he has to do the same process again and again. This leads to waste of time and resources. To optimize this, we have got standard workspace.

5.1 Standard Workspacce

The Standard workspace contain different client projects. We can work on one project and can easily switch to another whenever necessary. We can easily flash and validate different test cases. This reduces the time required for testing as well as the resource needed. This has saved more than an hour for the testers. In case tester has made an error and had altered the code and corresponding test case, then he need not initialize everything from beginning, instead he can use this workspace to flash and run the same client project which he has altered. Thus, this workspace makes efficient use of resources as well as the time required to finish the tasks.

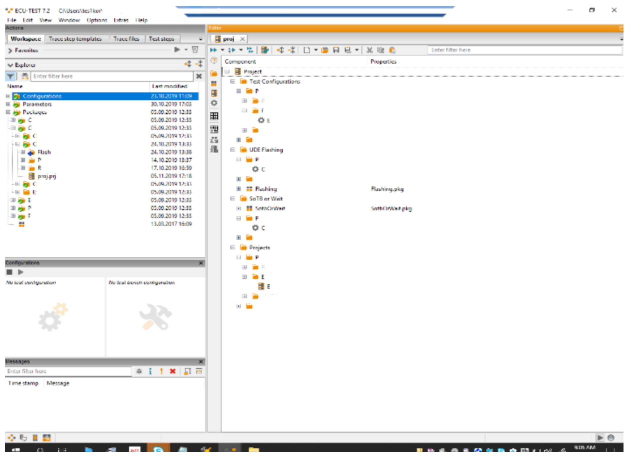


Figure 5.1 - Standard Workspace

5.2 Modified Standard Workspace

Modified standard workspace is more of user friendly as it doesn't need user to prefill the required information to run the project. Here initially when user run the project, he gets pop-up asking to select project. Once he does, it will automatically select the required project. It also sets the required CAN communication ID for that project to send the frames backend. Providing wrong CAN communication ID doesn't allow the information to be fetched. The tester doesn't have to worry about this here.

After selection of project it asks what type of communication is present in transport layer for the specific function. Once the user selects the required one, it will automatically select the required type of communication as well as the needed database information for this type of communication. Once this is set, it will interface with INCA and selects the same and provide the easy communication process and tester can start validating test cases or altering the variable in INCA through Ecu-Test and then testing on it.

In case user has provided wrong way of communication, it will ask again to provide the correct way of communication so that INCA interface happens without any issues.

This workspace includes BC regression packages. If tester wants to check this, he can enable the project. He will be asked to select what type of testing needs to be check. Once he selects the required ones, it will run only those tests. If tester doesn't respond within the given time, it selects all the test package and provide the corresponding result.

This workspace has saved more than an hour in setup run and an hour in process run. Thus, the new model is more efficient than previous workspace.

As there is cost involved in effort or time given for tester. This helps both tester and the company. The OEMs will get the required project in the earliest.

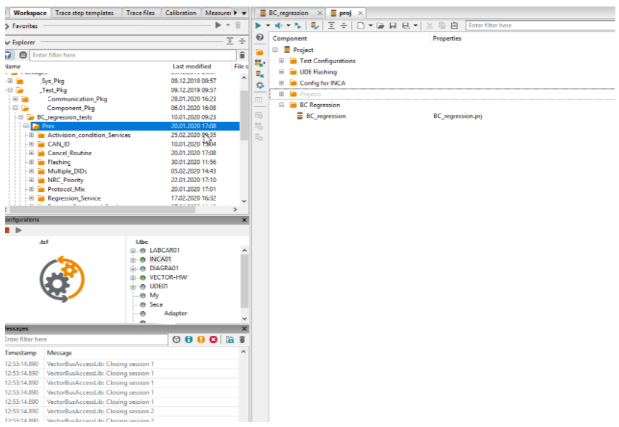
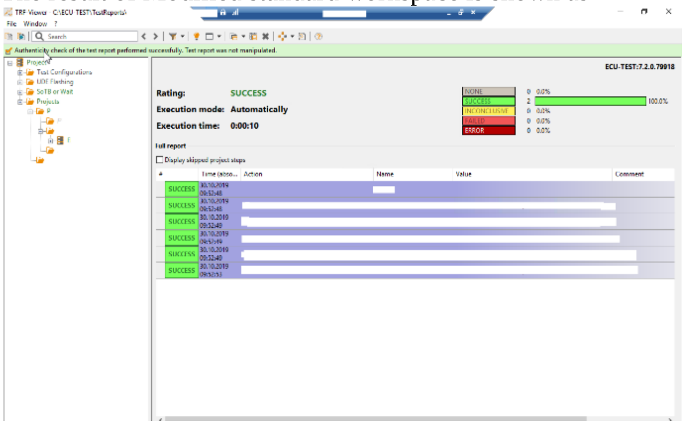
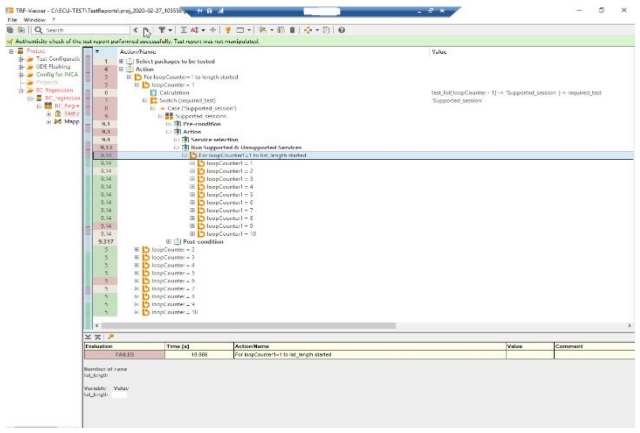


Figure 5.2 – Modified Standard Workspace with Basic Component regression

The result of Modified standard workspace is shown as





```
TRF-Viewer - C:\ECU-TEST\TestReports\proj_2020-02-27_105558
File  Window  ?
Authenticity check of the test report performed successfully. Test report was not manipulated.

Project                 Action/Name                                              Value
  Test Configurati    1     Select packages to be tested
  UDE Flashing        4     Action
  Config for INCA     5       For loopCounter=1 to length started
  Projects            5         loopCounter = 1
  BC Regression       6           Calculation                                    test_list[loopCounter - 1] -> 'Supported_session' |-> required_test
    BC_regressi       7           Switch (required_test)                         'Supported_session'
      BC_Regre        8             Case ('Supported_session')
        Test c        9               Supported_sessions
        Mapp          9.1               Pre-condition
                      9.1               Action
                      9.4                 Service selection
                      9.13                Run Supported & Unsupported Services
                      9.14                  For loopCounter1=1 to list_length started
                      9.14                    loopCounter1 = 1
                      9.14                    loopCounter1 = 2
                      9.14                    loopCounter1 = 3
                      9.14                    loopCounter1 = 4
                      9.14                    loopCounter1 = 5
                      9.14                    loopCounter1 = 6
                      9.14                    loopCounter1 = 7
                      9.14                    loopCounter1 = 8
                      9.14                    loopCounter1 = 9
                      9.14                    loopCounter1 = 10
                      9.317             Post-condition
                      5         loopCounter = 2
                      5         loopCounter = 3
                      5         loopCounter = 4
                      5         loopCounter = 5
                      5         loopCounter = 6
                      5         loopCounter = 7
                      5         loopCounter = 8
                      5         loopCounter = 9
                      5         loopCounter = 10

Evaluation | Time [s] | Action/Name                               | Value | Comment
FAILED     | 18.866   | For loopCounter1=1 to list_length started |       |

Number of runs
list_length

Variable    Value
list_length
```

Figure 5.3 - Results of Modified Standard Workspace with Basic Component regression

VI. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

As the demand for more features in automobile by customer in increasing and with the increasing number of tasks, there is increase in the complexity and thus automation is key in reducing the testers work and time. Thus, the modified standard workspace has helped the tester to reduce the time taken to setup the project as well as the time required for validation.

6.2 Future Scope

The Automation can be further improved by making all the results to be uploaded to the required location such as artifacts or provide notification in mail regarding the finish of validation.

The requirements from OEMs can be made to be in specific format so that after feeding the requirement, it can generate all the test cases automatically in the tool which can reduce the time required to write new test cases.

REFERENCES

1. Wang Tianxu and Gong Mingde proposed “Design electronic control unit of blend brake system for heavy vehicle” 2011 International Conference on Electronic & Mechanical Engineering and Information Technology.
2. E Vargil Vijay, Ch V Rama Rao, E Vargil Kumar & G N Swamy proposed “Electronic control unit for an adaptive cruise control system & engine management system in vehicle using electronic fuel injection” 978-1-4244-9005-9/10©2010 IEEE.
3. An Xiaohui, Liu Bolan, Cui Tao,Xie Zhenxing and Zhang Fujun proposed “Electronic Control Unit Development for Unit Pump Diesel Engine” 2010 International Conference on Optoelectronics and Image Processing.
4. Der-Cherng LIaw, Cheng-Yu Yu and Kuo-Chen Wu presented a paper “A CAN-Based Design for the Control of Electric Vehicle” 2014 14th International Conference on Control, Automation and Systems (ICCAS 2014) Oct. 22-25, 2014 in KINTEX, Gyeonggi-do, Korea.
5. Peng Hehuan proposed “Simulated Analysis on the Automobile Body Electrical Control System Based on CAN-LIN Bus” 2011 Third International Conference on Measuring Technology and Mechatronics Automation.
6. Parag Kharche, Meera Murali and Geetanjali Khot proposed “UDS Implemetation for ECU I/O Testing”
7. Sundaravadivelu K, Shantharan G, Prabharan P and Raghavendra N proposed “Analysis of vehicle dynamics using co-simulation of AVL-CRUISE and CarMaker in ETAS RT environment”.
8. Anca LUPEI and Loredana STANCIU proposed “Application for UDS Automated Test Generation” 11th IEEE International Symposium on Applied Computational Intelligence and Informatics, May 12-14, 2016, Timişoara, Romania.
9. Muneeswaran. A., proposed "Automotive diagnostics communication protocols analysis kwp2000, can, and uds," in IOSR Journal of Electronics and Communication Engineering, IOSR, Feb 2015, pp. 20-31.
10. Sooyong Jeong, Yongsub Kwak and Woo Jin Lee proposed “Software-in-the-Loop Simulation for Early-Stage Testing of AUTOSAR Software Component” Eight International Conference on Ubiquitous and Future Networks, 2016.

AUTHORS PROFILE

I’m **Shailendra S**, pursuing my M.Tech in VLSI Design and Embedded Systems in Bangalore Institute of Technology, Bengaluru. Recently, I’ve completed my internship in Robert Bosch in the field of automation.
I’ve completed my B.Tech in the field of Electronic and Communication Engineering in N.M.A.M Institute of Technology, Nitte. Our final project was named “WeeDCreWa” which is a prototype of an automated boat which helps to keep the purity of lake from getting spoiled by weeds. We got top best project award for it. I’ve done some mini projects on light sensor RC car, fire alarm, finger print door locker.

Prof. Vijaya Prakash A.M Graduated B.E from University Visveswaraya College of Engineering Bangalore of Bangalore University in the year 1992, Post graduated in M.E from SDMCET Dharwad of Karnataka University in the year 1997 and Ph.D.degree from Dr. M. G. R Educational and research institute University from Chennai. He has been actively guiding PG and UG student’s projects in the area of VLSI Design and Image Processing. He has 24 technical paper publications in international journals and international conferences. He is guiding 6 Ph.D scholars and one Ph.D degree is awarded under his guidance. Currently he has been working as Professor in Electronics and Communication Engineering Department, Bangalore Institute of Technology Bangalore. He has presented paper in Singapore. His research interests are Low Power VLSI, Image Processing. He is a member of IMAPS and ISTE and fellowship of Institute of Engineers India.

Myself **Nishanth Ravindranath**, Specialist working at RBEI (subsidiary of Bosch). I have 8 years of experience in the field of Tester Diagnostics, Flash Programming for Engine Control Units in Automobiles. My specialization also include Over the Air Update of Firmware using latest cryptographic algorithms. I am working on the automation of Tester Diagnostics for French OEMs which will ramp up the development and time to market which will be bring cost benefits for OEMs. This automation will act as continuous deployment approach which will help OEMs to add new features and deploy them quickly into the vehicles.