Low Cost Hardware In The Loop (HIL)

Test Tool

Luis E. Castaneda-Trejo  
The Electrical and Computer Engineering Department  
The University of MichiganDearborn, Michigan USA  
luisct@umich.edu

*Abstract*—Test and Validation teams across several automotive companies use COTS (Commercial Off-the-Shelf) technology for the design and development of Automated Test Equipment (ATE). During System Validation and Verification (V&V) phases, automotive R&D teams budget high costs for professional development tool licenses like Vector CANoe/CANalyzer, Vehicle Spy from Intrepid Control Systems or similar to execute test cases to vehicle Electronic Control Units (ECUs) under test. While these tools are excellent to design and develop large simulations and tests scenarios, once the design is finished, sometimes they are no longer needed. In this project, I used a low-cost microcontroller platform that can execute test cases to ECUs via CAN protocol commanded by an instruction received by a TCP client.

Keywords—Hardware-in-the-loop, Automated Test, Ethernet, CAN.

# Introduction

Commercial of the Shelf Technology (COTS) offer several solutions out of the box for automotive communications. Companies like Vector Informatik [1] or Intrepid Control Systems (ICS) [2] have specialized hardware and software tools to simulate complete Electronic Control Units (ECUs). Some of these commercial tools have become a standard in the automotive industry.

In R&D disciplines, most of System Validation and Verification teams rely on these types of tools to design and develop Automated Test Equipment (ATE) to communicate and execute test cases. Some of the benefits they offer are tool standardization, database homogenization and system model reuse from different projects.

These tools are excellent to design and develop large vehicle simulations and tests scenarios but once the test modes have been designed, users of these tools still need to have expensive runtime licenses to execute their developed models or continue using their development licenses impeding other team members to use them for their component development. Sometimes these models or internal releases are just for Proof-of-Concepts, test demos or custom implementations that make it difficult to justify the purchase of a high-cost development or runtime tool license.

The purpose of this Hardware-in-the-loop (HIL) Test Tool is to allow users implement and execute test scripts using CAN protocol to automotive ECUs without the need of expensive runtime or development licenses.

This project uses a development platform from ST Microelectronics [6] which has a low cost but highly capable microcontroller unit (MCU). This MCU can communicate with any ECU via Controller Area Network (CAN) and execute user defined test scenarios.

The user communication to the HIL Test Tool is via Ethernet, the HIL Test Tool has a TCP server so any TCP client can communicate with it and send command instructions to the Device Under Test (DUT).

This project was designed and built using the waterfall process methodology. The overall design of the system suits this development process well because the requirements are known and they did not change. Requirements, design, integration, and test phases were implemented and will be discussed in the following sections.

*Figure 10* in section V describes the system development process in the 3 main sections: Hardware, Software and Testing.

# Prior Work

Research work was conducted to find existing or alternate solutions for this problem. In Mutlu et al [3], authors show a test system created for design verification that is able to interact with the Device Under Test (DUT) and its sub-systems via RS422, RS485, Telnet and CANopen protocols but it lacks Ethernet communication which allows the user to execute test cases remotely. A more concrete and current commercial tool for the automotive industry would be the recently launched RAD-Meteor [4] from ICS which is an ethernet based tool with a similar concept than the proposed tool in this project but its focused purely on automotive ethernet ECUs.

Pintaric et al [5] proposes a framework of a flexible HIL interface utilizing Typhoon HIL and Siemens Amesim software and hardware for full vehicle emulation. This is another example of a high cost solution which works great for the mentioned market niche (Hybrid Electric Vehicles) but requires licensing for operation.

# Concept

The general purpose of this project is to emulate the functionality of an Automated Test Equipment (ATE) capable of running test scenarios via CAN to any type of ECU that has CAN communication available.

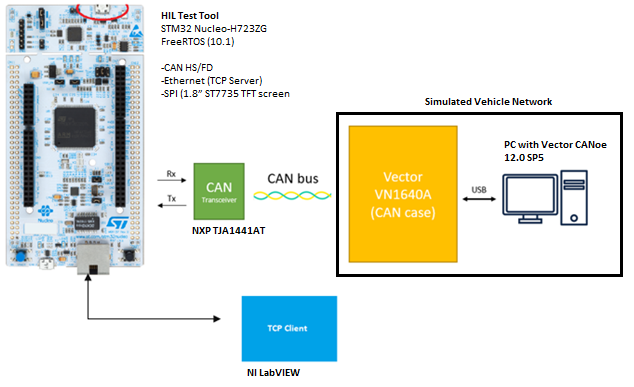


Figure 1. Project Elements: HIL Test Tool, Simulated CAN network and TCP client.

The test scenarios are commanded and sent by a TCP client. The tool has 4 predefined test scenarios that are mentioned in the requirements section and in Table 5. To have a Device Under Test to test the HIL Tool, a basic vehicle CAN network consisting of 3 ECUs was simulated using Vector CANoe [7].

The HIL Test Tool uses FreeRTOS [8] as operative system to handle the different tasks (application code) to interact with the ECUs implemented in the simulated network in CANoe. The application code has a total of seven tasks. Four of them will execute a predefined Test Mode. The software architecture is modular so in case there is a need to add more test modes, the design pattern supports the addition of new tasks just by adding them to *freertos.c* via the Integrated Development Environment (IDE) which in this case was STM32CubeIDE [14].

*Figure 1* shows the project components. Section IV describes the different requirements for the main project components.

The yellow box in *Figure 1* represents the CAN interface to allow physical devices to interact with the simulated network. Any TCP client can interact with the HIL Test Tool, a custom TCP client was developed using NI LabVIEW [9] to have a better interaction with the Test Tool. For demonstration purposes the Hercules Utility Tool [11] provides additional functionality to interact with Test Tool so it was the preferred method used during the Integration and Testing development phase.

# REQUIREMENTS

In this section, the system requirements will be described. The overall solution has 3 main groups: HIL Test Tool, CAN Network and TCP client.

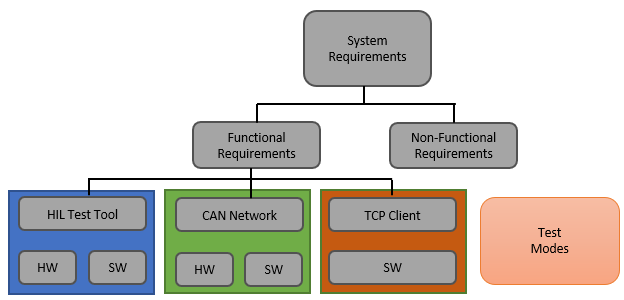


Figure 2. Overall System Requirements.

Although the complete demonstration includes the 3 mentioned groups, the rest of the sections will focus mainly on the HIL Test Tool.

## HIL Test Tool Requirements

| ID | HIL Test Tool Hardware Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| HW-001 | Dev. Board | Functional | Board has 3 CAN HS/FD controllers. |
| HW-002 | CAN Transceiver | Functional | NXP TJA1441AT is used as Tx. |
| HW-003 | Ethernet Comm. | Functional | Board has Ethernet connection. |
| HW-004 | CAN termination | Functional | 120Ohm resistor used as termination. |
| HW-005 | Ethernet cable | Functional | CAT6 cable is used. |
| HW-006 | CAN Connector | Functional | A DB9 connector is used for PINs 2 & 7. |
| HW-007 | CAN cable | Functional | A twisted pair cable is used for comm. |
| HW-008 | LCD screen | Functional | Adafruit ST7735 1.8” display. |

Table 1. HIL Test Tool Hardware Requirements.

| ID | HIL Test Tool Software Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| SW-001 | RTOS | Functional | FreeRTOS is used |
| SW-002 | CAN driver | Functional | Driver configured for 500kbaud |
| SW-003 | Ethernet driver | Functional | Driver for Ethernet Comm. |
| SW-004 | TCP Server | Functional | Adapt TCP Server library. |
| SW-005 | Global header lib | Functional | .h to allocate required variables. |
| SW-006 | Software Arch. | Functional | Modular & scalable. |
| SW-007 | Test Scripts | Functional | Modular & scalable. |
| SW-008 | CAN Msg Format | Functional | Set CAN msgs in UDS format. |

Table 2. HIL Test Tool Software Requirements.

*Figure 3* shows the global path that data follows when a command is received by the HIL Test Tool, how is it processed by the TCP handle process and how the CAN message is sent to the network.

Similarly, *Figure 4* shows the Data Flow Diagram for the inverse process when the ECU responds back with the information via CAN needed to evaluate and apply PASS/FAIL criteria.

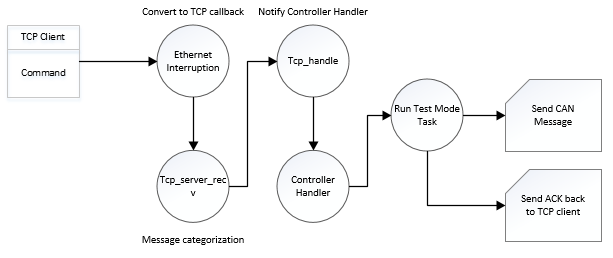


Figure 3. Data Flow Diagram for TCP message

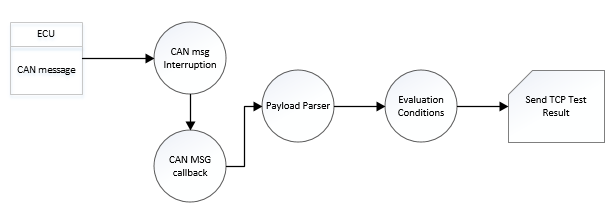


Figure 4. Data Flow Diagram for ECU message response

## Simulated CAN Network Requirements

This section describes the requirements for the implemented ECUs. Since the ECUs are only used to test the capabilities of the HIL Test Tool, their data diagrams on how the ECUs were implemented are not included in this report.

| ID | Simulated CAN Network Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| SM-001 | CAN network | Functional | CAN network with at least 1 ECU. |
| SM-002 | ECU1: Engine | Functional | Read/write Speed signal |
| SM-003 | ECU2: Lights | Functional | Read/write Light & hazard signal |
| SM-004 | ECU3: Display | Functional | Read/write panel values. |
| HW-009 | CAN Interface | Functional | Vector VN case 1640 |

Table 3. Simulated CAN network Requirements.

## TCP Client Requirements

This section describes the requirements for the TCP client. The TCP client was only used to test the capabilities of the HIL Test Tool, their data diagrams are not included in this report.

| ID | TCP Client Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| TCP-001 | TCP Client: Open Conn | Functional | Opens TCP connection. |
| TCP-002 | TCP Client send command | Functional | Send a command. Example: Test Mode 1 |
| TCP-003 | TCP Client: Close Conn | Functional | Closes TCP connection. |

Table 4. TCP Client Requirements

## Test Mode Requirements

As seen in figure 2, the Test Mode requirements are a separate entity from the HIL test tool. This is expected because each project will have its own Test Plan and their own test requirements. For demonstration purposes a set of 4 test modes are included as part of this project.

| ID | Test Mode Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| TM-001 | Speed Engine | Functional | Verifies the speed set to the ECU. |
| TM-002 | Lights | Functional | Verifies the lights turn ON/OFF. |
| TM-003 | Hazards | Functional | Verifies the hazard lights turn ON/OFF |
| TM-004 | Engine status | Functional | Verifies the ignition status of the engine. |

Table 5. Test Mode Requirements.

## Non-Functional Requirements

The system has several non-functional requirements. One of them is the time it takes a command to reach the HIL Test Tool. This time can vary widely based on different aspects for example, the network (LAN) load. Another non-functional requirement is the need of an acknowledgment sent from the Test Tool back to the TCP client. The tool is designed to work with any TCP client that can send a string of characters. If the client can read this string, it will display the status of the tool.

# Project Elements

This section describes the parts of the project that were used both in hardware and software.

## Hardware – HIL Test Tool

The Test Tool hardware consists of a Nucleo-H723ZG which has an STM32H7 (Arm 32-bit Cortex-M7) with 1 Mbyte of Flash and 320 Kbytes of RAM. This board has access to one of the three available CAN controllers supporting Flexible Data (FD) rate. The CAN interface is configured as CAN High Speed (HS) only because the information required for the described test modes in *Table 5,* do not require more than 8 bytes of payload in the CAN message.

To communicate with a CAN network, the TJA1441AT [10] CAN transceiver from NXP was used. This transceiver supports up to 5 Mbit/s in FD mode. The configured speed for the CAN controller is 500 Kbytes.

To display the status of the Test Tool, a small 1.8” TFT screen (ST7735) was connected using SPI communication. The bus speed is set to 6 MBits/s. SCK signal is connected to PA5 and the Master Out Salve In (MOSI) signal is connected to PD7 of the development board.

## Software – HIL Test Tool

Software in the ECU uses a Real-Time Operative System (FreeRTOS) to handle the tasks inside the project. Figure 5 shows the main software architecture. The project makes use of 2 external libraries. The TCP Server library and the TFT screen for SPI protocol were developed by ControllersTech.com [12][13].

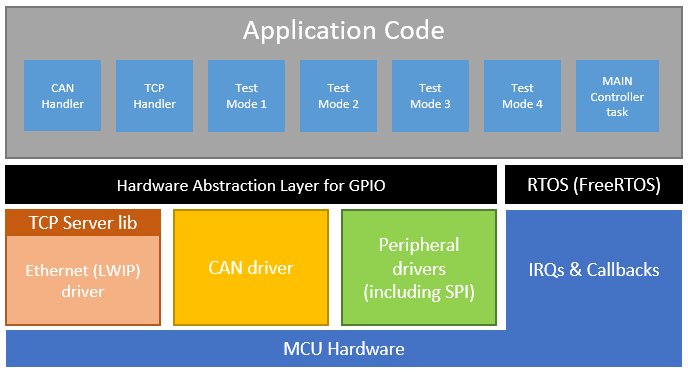


Figure 5. HIL Test Tool software architecture.

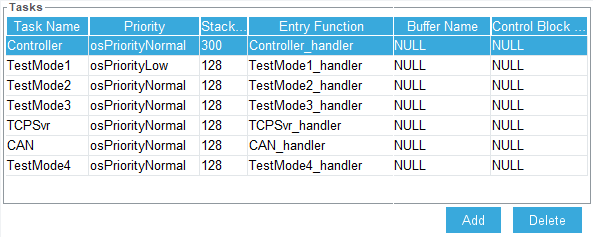
*Figure 6* show the tasks properties used in the project. Section V contains a detailed description of each task.

Figure 6. FreeRTOS tasks declared in STM32CubeIDE.

## Hardware – Simulated CAN Network

The simulated CAN network provides the right environment to test the HIL Test Tool. A VN1640A CAN case from Vector was used as interface to connect the tool to a CAN network. Any CAN interface would work as long as it responds to the CAN messages sent by the HIL Test Tool.

The VN1640A is a modular tool that supports CAN and LIN interfaces. For this project, CAN channel number 2 was used as the main interface. *Figure 7* shows the configuration applied to obtain a 500Kbyte speed network.

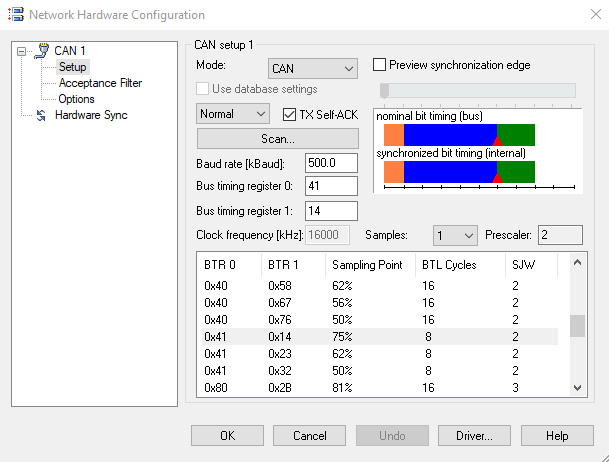


Figure 7. CAN interface configuration in CANoe.

*Figure 8* shows the mapping of the CAN channel number used to interface with the ECU in CANoe.

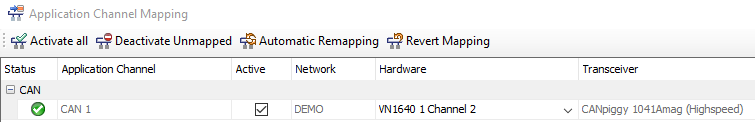
**

Figure 8. Physical CAN port mapping.

## Software – Simulated CAN Network (CANoe)

The simulated CAN network was implemented using Vector CANoe. CANoe is a commercial off-the-shelf software tool to develop, test and analyze individual ECUs and entire networks. It comes preloaded with examples to quickly start analyzing automotive networks.

The following CAN network was implemented based on one of the examples that came with the tool and was modified to show the data being sent to and from the HIL Test Tool. *Figure 9* shows the complete CAN network and the 3 ECUs (Engine, Light and Display) in it.

The Engine ECU handles the ignition status as well as the speed of the vehicle. The Light ECU handles the headlights and hazards of the vehicle, and the Display ECU handles the indicators of the panels showing the speedometer and the rest of the indicators in the panels.

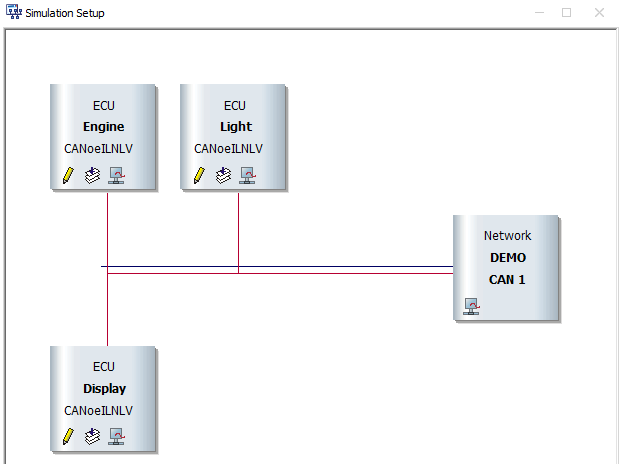


Figure 9. Simulated CAN Network.

# DESIGN AND IMPLEMENTATION

This section describes the design approach used for the project. *Figure 10* shows the Flow Diagram for a single test mode command.

The whole project has 3 sections: Hardware, Software and Test. *Figure 11* shows the flow diagram for each group of tasks in each section.

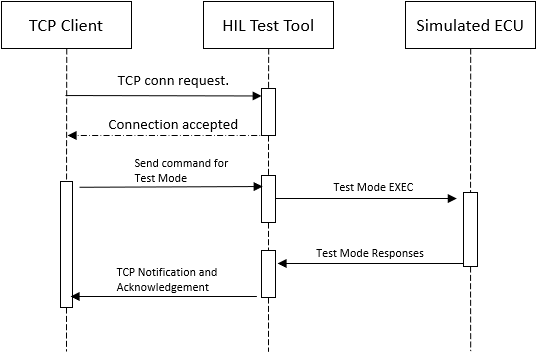


Figure 10. Flow Diagram for a single test mode command.

## Project Development Process

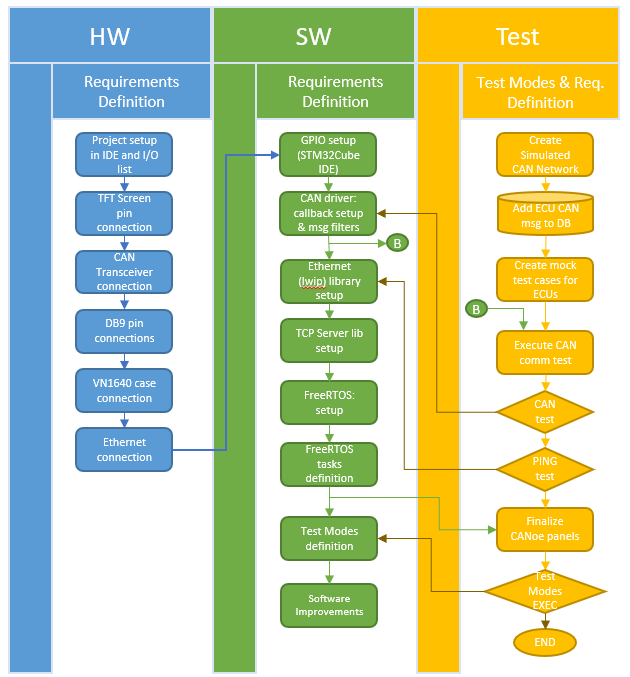


Figure 11. Flow Diagram for Task development.

The system development process chosen for the project was the waterfall model. The requirements definitions were static, all requirements were defined in the project proposal and approved so there was no need to have requirements modifications or reviews while the project was in progress. The waterfall model fits well to this scenario because each phase was completed before moving to the next one. The SDP Phase column in Table 6 shows the 4 phases of the project.

Once the hardware setup was completed and all devices connected, the next phase was the software design, in this phase as seen in *Figure 11,* the CAN driver, Ethernet, TCP Server and FreeRTOS were configured and integrated in the HIL Test Tool side, on the other hand the CANoe configuration was defined and panels, variables and CAN communication was enabled to receive initial test messages from the HIL Test Tool.

For the Integration and System Testing phase the Test Tool and the CAN network containing the ECUs were put together and verified. During this phase the HIL Test Tool was sending the corresponding CAN messages to the ECUs and verified that they were received in CANoe. The TCP client and Test Modes were also verified in this phase.

Finally in the Operation and Maintenance phase, system validation was performed, and some bugs were corrected.

Table 6 summarizes the above process. This table shows each section matching with its corresponding System Development Phase according to the waterfall model.

| ID | System Development Process (SDP) | | |
| --- | --- | --- | --- |
| Phase | Section | Activity |
| RQ-001 | Req. Definition | Hardware (HW) | Component list, devices, and connection definition. |
| RQ-002 | Req. Definition | Software (SW) | Software and task requirements. |
| RQ-003 | Req. Definition | Test (TST) | Test Mode definition and coverage. |
| SYS-001 | System (SYS) Design | Hardware (HW) | Project setup in IDE and I/O definition. |
| SYS-003 | System (SYS) Design | Hardware (HW) | TFT Screen pin connection. |
| SYS-004 | System (SYS) Design | Hardware (HW) | CAN Transceiver connection. |
| SYS-005 | System (SYS) Design | Hardware (HW) | DB9 pin connection. |
| SYS-006 | System (SYS) Design | Hardware (HW) | VN1640 case connection. |
| SYS-007 | System (SYS) Design | Hardware (HW) | Ethernet connection. |
| SYS-008 | System (SYS) Design | Software (SW) | GPIO setup and const name creation in IDE. |
| SYS-009 | System (SYS) Design | Software (SW) | CAN driver configuration, callback setup and message filters creation. |
| SYS-010 | System (SYS) Design | Test (TST) | Create simulated CAN network in CANoe. |
| SYS-011 | System (SYS) Design | Test (TST) | Create and add test CAN messages to database in CANoe. |
| SYS-012 | System (SYS) Design | Test (TST) | Create mock test cases for ECUs. |
| SYS-013 | System (SYS) Design | Software (SW) | Configure Ethernet communication in IDE and generate code. |
| SYS-014 | System (SYS) Design | Software (SW) | Integrate TCP Server library. |
| TST-001 | Integration and Testing | Software (SW) | Perform basic IP address from TCP client to HIL Test Tool. PING test. |
| SYS-015 | System (SYS) Design | Software (SW) | Setup FreeRTOS and task creation (Application Code). |
| TST-002 | Integration and Testing | Software (SW) | Created Test Mode code |
| TST-003 | Integration and Testing | Software (SW) | Executed Test Modes |
| OP-001 | Operation & Maintenence | Software (SW) | Software improvements |
| OP-002 | Operation & Maintenence | Software (SW) | ECU communication improvements |
| OP-003 | Operation & Maintenence | Document (DOC) | User Manual |

Table 6. Summary of System Development Process.

| IDs | Project Progress | | | |
| --- | --- | --- | --- | --- |
| Phase | Section | Date | Status |
| RQ-001, RQ-002, RQ-003 | Req. Definition | HW. SW & TST | 9/23/2022 | Release |
| SYS-001, SYS-002,  SYS-003, SYS-004, SYS-005, SYS-006, SYS-007 | System (SYS) Design | HW | 10/09/2022 | Release |
| SYS-008, SYS-009 | System (SYS) Design | SW | 10/15/2022 | Release |
| SYS-010,  SYS-011,  SYS-012 | System (SYS) Design | TST | 10/16/2022 | Release |
| SYS-013,  SYS-014,  SYS-015 | System (SYS) Design | SW | 11/05/2022 | Release |
| TST-001,  TST-002,  TST-003 | Integration & Testing | SW | 11/15/2022 | Release |
| OP-001,  OP-002 | Operation & Maintenence | SW | 11/22/2022 | Release |
| OP-003 | Operation & Maintenence | DOC | 11/25/2022 | Release |

Table 7. Project progress per phase.

The following sections will describe the software parts implemented in the HIL Test Tool. The parts are CAN, Ethernet & TCP Server, GPIO, RTOS & Application Code and Interruptions.

## HIL Test Tool Software

### Controller Area Network (CAN)

The Nucleo-H723ZG has 1 CAN controller with FD support. This project uses the controller as CAN High Speed. *Figure 12* shows the most important parameters to achieve 500 kb speed with an input clock speed of 40Mhz.

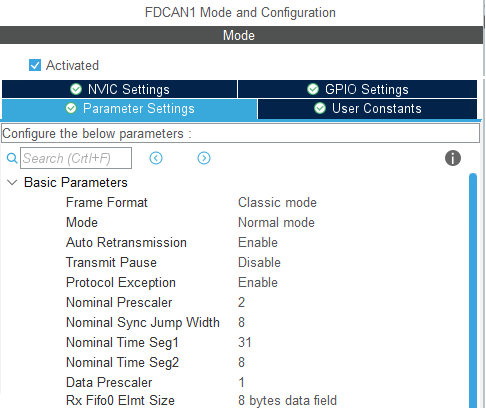


Figure 12. CAN basic configuration.

*Figure* 13 shows the calling graph of the CAN feature. The right side shows the callback function that gets triggered when a CAN IRQ is received and matches the identifier filter, if a CAN message is received but its identifier is different than the one registered then it is ignored. After the CAN message passes the identifier filter then the callback function passes the information to its corresponding test mode handler (task) via a FreeRTOS notifier.

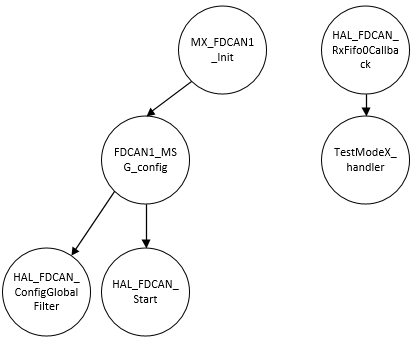


Figure 13. CAN communication Calling Graph.

### CAN Message Format

Most ECUs in the automotive industry rely on diagnostics for validation and verification purposes. This project sets the CAN frames similar to the format used in the Unified Diagnostic Services (UDS) [15]. Table 7 shows the format of the CAN messages used in the project.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Byte #** | **Request (0x762 to ECU)** | **Data/Range (HEX)** | **Resp (0x726 from ECU)** | **Data/Range (HEX)** |
| 0 | # of bytes sent to the ECU (excluding this byte) | 05 | # of bytes sent from the ECU (excluding this byte) | 05 |
| 1 | Service ID (SID) | 22 | Positive response to SID. (SID + 0x40) | 62 |
| 2 | Diagnostic Service (ex. 0xFE01 = Speed Engine) | $FE | Same as Request | $FE |
| 3 | $01 | $01 |
| 4 | Service Enable | 00 – 01 | 00 – 01 |
| 5 | Service data byte 1 | 00 – FF | 00 – FF |
| 6 | Service date byte 2 | 00 – FF | 00 – FF |
| 7 | Not Used | 00 | 00 |

Table 7. CAN Message format

### Ethernet/TCP Server

The Nucleo-H723ZG has 1 ethernet controller. This board has a Reduced Media Independent Interface (RMII) with access to memory configuration.

Figure 13 shows the most important parameters to configure the Ethernet correctly. There are more parameters that were configured. These parameters are in the lightweight IP section of the STM32CubeIDE project.

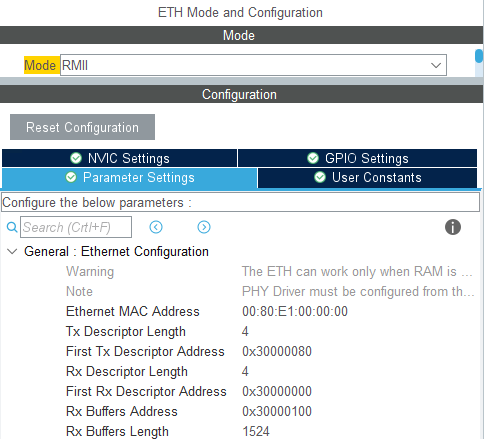


Figure 13. Parameter settings for MCU Ethernet address

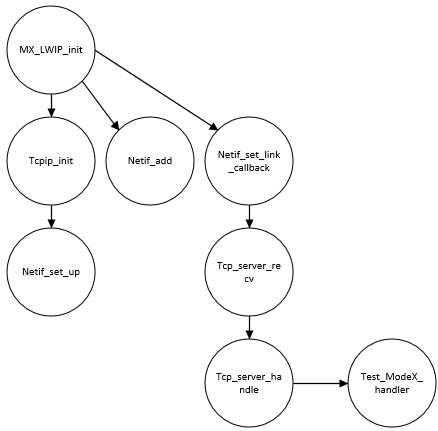


Figure 14. Ethernet and TCP Server Calling Graph.

The middleware TCP/IP stack is the lightweight IP (lwIP) which is an open-source stack intended for embedded devices. A special package is provided for each STM32 MCU series. The lwIP includes an Ethernet Hardware Abstraction Layer (HAL) with support for RTOS like FreeRTOS.

Figure 14 shows the calling graph to initialize the lwIP layer, port configuration and IP address to start sending commands to the simulated ECUs via CAN.

### GPIO

Table 7 shows the I/O pins used for the project. The two devices connected to the board were the TFT screen and the CAN transceiver. Column three of *Table 7* show to which device each pin belongs to.

| PIN | GPIO | | |
| --- | --- | --- | --- |
| Name | Device | Description |
| PA5 | SPI1\_SCK | TFT screen | SPI clock |
| PD7 | SPI1\_MOSI | TFT screen | Main Output Sub node In |
| PC0 | CS | TFT screen | Chip select. ON |
| PA3 | D/C | TFT screen | Display data/command selection |
| PD0 | FDCAN1\_RX | CAN transceiver | Rx line for MCU |
| PD1 | FDCAN1\_TX | CAN transceiver | Tx line from MCU |

Table 7. GPIO list

### RT Operative System and Tasks (Application Code)

The chosen OS was FreeRTOS. FreeRTOS is a market-leading real-time operative system (RTOS) for microcontrollers and small microprocessors. It is a lightweight RTOS, allows modularization of tasks to create application code and is a good option to have it as a framework for this project.

A total of 7 RTOS tasks were created. Controller handler, CAN handler, TCP handler, Test Mode 1 handler, Test Mode 2 handler, Test Mode 3 handler and Test Mode 4 handler . All the tasks are in freertos.c file within the project. The following sub sections will describe each task in more detail.

#### Controller Handler

This is the main task of the application code. It has 2 states: INIT and IDLE. The INIT case initializes the lwip layer, the TCP server, the TFT screen, will print out the title, software and hardware versions of the system in the screen. Once the initialization state finishes, it will move to the IDLE case. The IDLE case will wait until the TCP Server (tcpServerRAW.c) sends a notification that a new command has been received.

Depending on the received message, the IDLE case will call the corresponding test mode handler (task) and go back to listening mode to wait for another incoming message. Table 8 shows the functions contained by the Controller Task.

| State Name | Controller handler task | | |
| --- | --- | --- | --- |
| Function | Parameters | Description |
| INIT | MX\_LWIP\_Init | None | Initialize lwip. |
| INIT | Tcp\_server\_init | None | Initialize TCP server. |
| INIT | ST7735\_Init | Degree rotation | Initialize TFT screen in horizontal mode. |
| INIT | ST7735\_WriteString | Coordinates and text to print | Prints program title. |
| INIT | HAL\_FDCAN\_AddMessageToTxFifoQ | Address of CAN1, Identifier and payload | Sends an initial CAN msg to the DUT |
| IDLE | xTaskNotifyWait | Time to wait, test mode address. | Waits test mode from TCP client. |

Table 8. Controller handler task function states.

#### CAN Handler

The CAN handler receives a notification from the CAN interrupt that a matching CAN frame with an identifier of 0x762 has been received. If the interrupt receives an 0xFF in byte #4 it sets the DUT\_FAILURE flag to ON, indicating that the device under test sent a Negative Request Code (NCR), otherwise the whole message gets copied to a myRxData which is a global variable ready to be processed by the CAN handler. Once the notification is received by the CAN handler task it sends a notification to the corresponding Test Mode handler task indicating that a specific test mode shall be executed. Table 9 shows the functions contained by the CAN Handler Task.

| Function Name | CAN handler task | | |
| --- | --- | --- | --- |
| Input | Output | Description |
| xTaskNotifyWait | Flag address, ticks to wait | Status | Waits for calling task to notify a CAN msg is ready |
| HAL\_FDCAN\_AddMessageToTxFifoQ | CAN1 controller address., identifier and msg payload. | HAL\_OK | Sends a CAN msg to DUT |
| xTaskNotifyWait | Bits to clear on entry, Ticks to wait. | status | Waits for the CAN interrupt task to receive a notification. |
| xTaskNotify | Test Mode handler | Not used | Sends an event directly to unblock an RTOS test mode task. |

Table 9. CAN handler task function

#### Test Mode Handlers

There are 4 test modes included in this project. The test modes are defined in Table 5 of the requirements section. The recommended structure for each test mode is shown in Figure 15. The initialization (INIT) state should contain the required DUT configuration i.e., enter a specific diagnostic mode or a special configuration for the test cases defined in the MAIN section to run. Finally in the CLEANUP state, all references should be closed, and the DUT should return to normal mode.

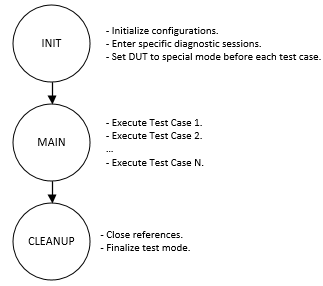


Figure 15. Test mode structure.

The user can add new Test Modes by clicking the add button shown in Figure 6 or manually copying and renaming one the available Test Modes and declaring the task with its parameters at the beginning of *freertos.c*. The stack size for each test modes is 128 kb but can be incremented if needed.

### Interruptions

The hardware interruptions (IRQs) are to determine if a new event triggers a CAN or TCP operation. The HIL Test Tool has 2 main callbacks that happen after one of the mentioned interruptions occur: *HAL\_FDCAN\_RxFifo0Callback* and *tcp\_server\_recv.* Figure 12 and Figure 14 show the Data Flow Diagrams of these functions.

## Simulated CAN network and TCP client

In order to test the capabilities and functions of the HIL Test Tool a simulated CAN network was implemented in CANoe. CANoe comes with several examples of different types out of the box. The *easy.cfg* configuration was used as baseline and adapted to meet the requirements of the Test Tool.

# Results

The metrics that were used to validate the requirements of this project were: System able to show PASS/FAIL criteria, Data validation and Test case modularity. The following will present the result for the 3 cases.

## PASS/FAIL Criteria

Using the TFT screen, the Test Tool is able to show the PASS/FAIL status of each Test Mode. The nucleo board also has 3 on-board LEDs that indicate the status too. The RED led turns on when there is a failure detected in the CAN message payload, the GREEN led turns on when there are no Negative Codes in the CAN message and the YELLOW led turns on when a test mode is being executed. Figure 16 shows a sample of the fail criteria.



Figure 16. Failed message sample.

## Data Validation

The validation of the transmitted data to the DUT was verified using CANoe as seen on Figures 17 and 18. CANoe ran the simulated ECUs but also allows the monitoring of the whole CAN network so the Data Window and Graph Window show the data being received correctly. Each Test Mode has a delay time of 10 ms per cycle but can but can changed to match a new test mode which could require a faster or slower execution time.

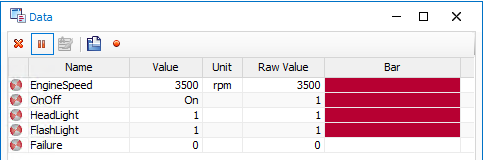


Figure 17. CAN Data Window.

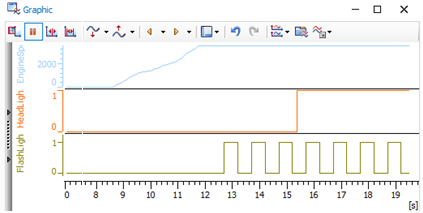


Figure 18. CAN Graph Window.

## Test Case Modularity

Adding new Test Modes is an easy process. User needs to add a new task using the IDE as shown in Figure 6 and using the Sate Machine design pattern from the other Test Mode tasks. The recommended test mode state diagram is shown in Figure 15. The recommended task priority is NORMAL so that priority inversion does not occur during operation.

# CONCLUSIONS

This project sets a good baseline tool and framework for test equipment engineers or system test engineers. All requirements were met, and a good modular architecture was achieved allowing users to add more test cases/modes or adapt the ones in the system to tailor their needs. The HIL Test Tool does not need any special licenses, so it provides great cost-value as a system testing tool compared with other commercial tools. Something to notice is that while the STM32 Nucleo platform is great to develop project ideas and proof of concepts it is an evaluation board and is not intended for final product production. The cost would be too high, and board availability is not guaranteed. A custom board would have to be created.

A second thing to notice is that the C programming language is not the most used language by test engineers and usually these teams need a scripting rather than a development language so a platform like the Raspberry Pi, Beagle Bone or similar supporting a high level language like Python would be a better fit mainly because the mentioned platforms are computers with bigger CPUs, more memory and external storage that MCU boards like the Nucleo platform don’t have.

But in case that a high determinism is required as part of the test requirements then platforms like the ones described in the project would excel in performance.

##### References

1. Vector Informatik (www.vector.com/us/en/company/about-vector/)

[2] Intrepid Control Systems (<https://intrepidcs.com/>)

[3] H. Mutlu, H. C. Atakan and Y. Yıldırım, "System Level Design Verification Testing for Different Interconnected units with Hardware-in-the-loop (HIL) Simulation," 2019 IEEE AUTOTESTCON, 2019, pp. 1-3, doi: 10.1109/AUTOTESTCON43700.2019.8961890.

[4] RAD Meteor – 10BASE-T1S (<https://intrepidcs.com/products/automotive-ethernet-tools/rad-meteor/>)

[5] I. Pintaric et al., "Flexible HiL Interface Implementation for Automotive XiL Testing," 2021 IEEE Vehicle Power and Propulsion Conference (VPPC), 2021, pp. 1-3, doi: 10.1109/VPPC53923.2021.9699257..

[6] Nucleo H723ZG ([www.st.com/en/evaluation-tools/nucleo-h723zg.htm](http://www.st.com/en/evaluation-tools/nucleo-h723zg.htm)

[7] CANoe (<https://www.vector.com/int/en/products/products-a-z/software/canoe/>)

[8] FreeRTOS (<https://www.freertos.org/>)

[9] What is LabVIEW? (<https://www.ni.com/en-us/shop/labview.html>)

[10] NXP CAN Transceiver (<https://www.nxp.com/docs/en/data-sheet/TJA1441.pdf>)

[11] HERCULES TCP Utility Tool (<https://www.hw-group.com/software/hercules-setup-utility>)

[12] TCP Server Library (https://controllerstech.com/stm32-ethernet-4-tcp-server/)

[13] TFT display library (<https://controllerstech.com/interface-tft-display-with-stm32/>)

[14] STM32CubeIDE (<https://www.st.com/en/development-tools/stm32cubeide.html>)

[15] Unified Diagnostic Services (<https://www.csselectronics.com/pages/uds-protocol-tutorial-unified-diagnostic-services>)