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 programs 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 a vehicle Electronic Control Unit (ECU) 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. This project uses a low-cost microcontroller platform that can execute specific test cases to an ECU using 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 or Intrepid Control Systems 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, system model reuse from software development teams among others.

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. Sometimes these models are for Proof of Concepts purposes, test demos or small 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 test developers to implement and execute their already developed test scripts without the need of Runtime licenses.

This project uses a development platform from ST Microelectronics which has a low cost but highly capable MCU. This MCU can communicate with any ECU via CAN and execute user defined test scenarios. The user communication to the HIL Test Tool is via Ethernet, the HIL Test Tool acts as 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 deign well because the requirements are known and for demonstration purposes they will not change. A requirements phase, design phase, implementation phase and test phase were implemented and will be discussed in the following sections

The test modes in the other hand, were developed in sprints using Test Driven Development. Section XYZ describes the system development progress in the 3 main sections: Hardware, Software and Testing.

# Concept

The general purpose of this project is to emulate the functionality of an Automated Test Equipment (ATE) capable of running pre-defined test scenarios via CAN communication to any type of ECU that has CAN communication available. For this project, a basic vehicle CAN network consisting of 3 ECUs is simulated using Vector CANoe to demonstrate the functions of the HIL Test Tool.

The HIL Test Tool uses FreeRTOS as operative system to handle the different tasks (application code) to interact with the DUTs implemented in the simulated vehicle network in CANoe. The application code consists of 3 different tasks. Each task will execute a predefined Test Mode. The software architecture is modular so in case there is a need to add more test cases, the design pattern supports the addition of new tasks just by adding new tasks within freertos.c

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 but for this project a custom TCP client was developed using NI LabVIEW to have a better interaction with the Test Tool.

*Figure 1* shows the overall concept of the project and the main project elements. The next section will describe the different requirements for the main project components.

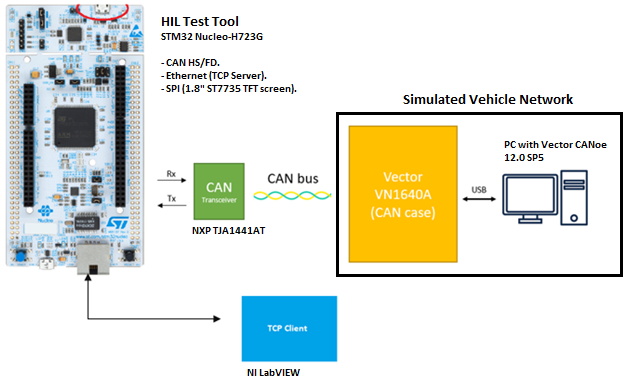
**

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

# REQUIREMENTS

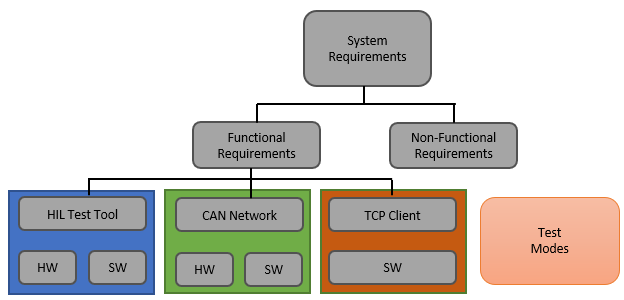
In this section, the system requirements for this project are shown and divided in 3 groups: HIL Test Tool, CAN Network and TCP client.

Figure 2. Overall System Requirements

## HIL Test Tool Hardware 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. |

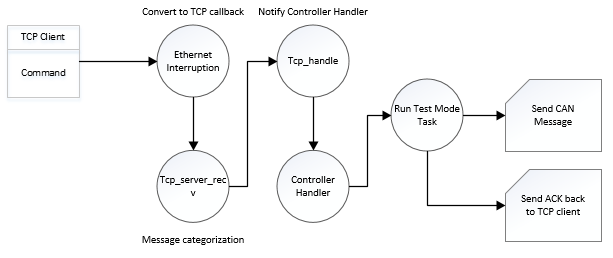
Table 1. HIL Test Tool Hardware Requirements.

| ID | HIL Test Tool Software Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| SW-001 | RTOS | Functional | FreeRTOS is used |
| SW-002 | CAN bus speed | Functional | 500kbaud is configured |
| SW-003 | Serial COM | Functional | UART1 enabled |
| SW-004 | RT response | Functional |  |
| SW-005 | TCP Comm. | Functional | Board has a TCP server. |
| SW-006 | Software Arch. | Functional | Modular & scalable. |
| SW-007 | Test Scripts | Functional | Modular & scalable. |

Table 2. HIL Test Tool Software Requirements.

Figure 3 shows the 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 7 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.



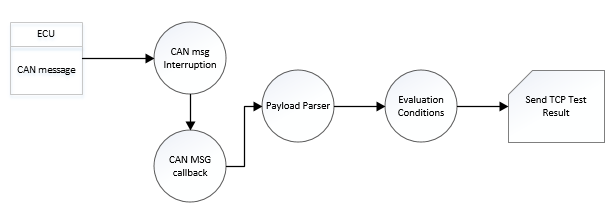
Figure 3. Data Flow Diagram for TCP message

Figure 4. Data Flow Diagram for ECU message response

## Simulated CAN Network Requirements

| 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

| ID | TCP Client Requirements | | |
| --- | --- | --- | --- |
| Name | Type | Description |
| TCP-001 | TCP Client UI | Functional | GUI design and developed in LV. |
| TCP-002 | Modular Design | Functional | Supports additional states. |
| TCP-003 | Known Design Pattern | Functional | State Machine based application. |

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 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 hazards 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 the load of the local area network (LAN). A second variable would be the acknowledgment sent from the Test Tool back to the TCP client. The tool is designed to work with any TCP client in the market that can send a string of characters. As long as the client is able to read a string of characters 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.

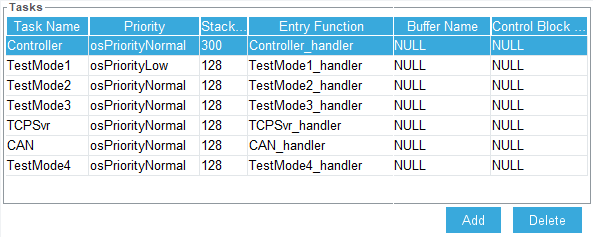
## Hardware – HIL Test Tool

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. The board has access to 1 CAN controller supporting flexible data rate. The CAN interface is configured as CAN High Speed (HS) only because the information required for the test mode does not require more than 8 bytes of payload.

To communicate with a CAN network, the TJA1441AT 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 protocol. The bus speed is set to 6 MBits/s. SCK signal is connected to PA5 and MOSI signal is connected to PD7 of the development board.

## 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 HIL Test Tool to the real CAN network.

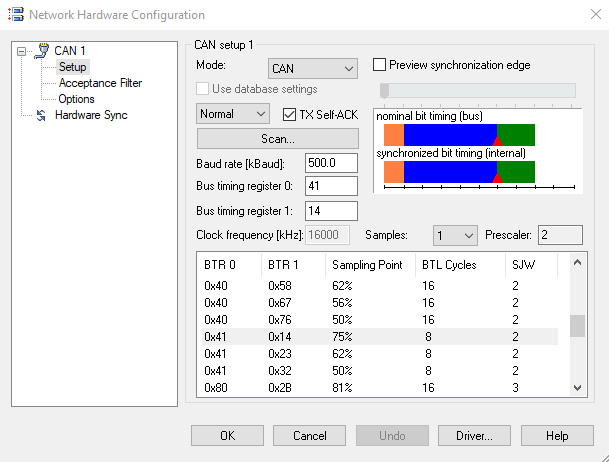
The VN1640A is a modular interface that supports CAN and LIN interfaces. CAN 2 channel was used as the CAN interface. The CANoe setup shown in Figure 5 was applied to achieve a 500Kbyte speed network.

Figure 5. CAN interface configuration in CANoe.

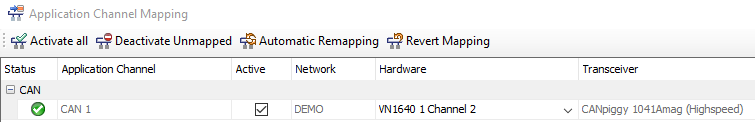
Figure 6 shows the mapping of the CAN channel number used to the interface with the ECU.

Figure 6. Physical CAN port mapping.

## Software – HIL Test Tool

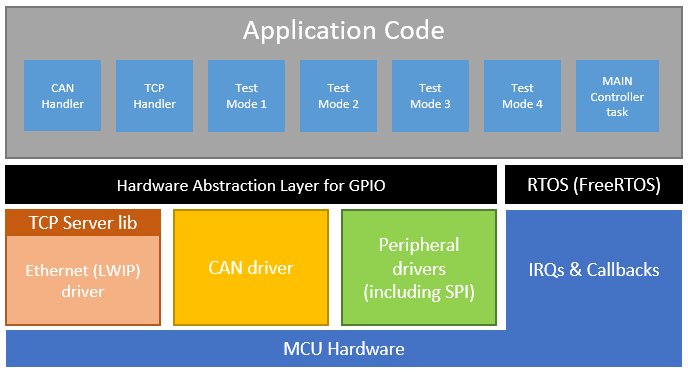
Software in the ECU uses a Real-Time Operative System (FreeRTOS) to handle the tasks of the project. Figure 7 shows the main software architecture.

Figure 7. HIL Test Tool software architecture.

Figure 8 summarizes the FreeRTOS tasks used in the project. Section V contains a detailed description of each task.

Figure 8. FreeRTOS tasks declared in STM32CubeIDE.

## 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 8 shows the complete CAN network and the 3 ECUs (Engine, Light and Display).

The Engine ECU handles the ignition status as well as the speed of the vehicle. The Light ECU is in charge of handling 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 of the panels.

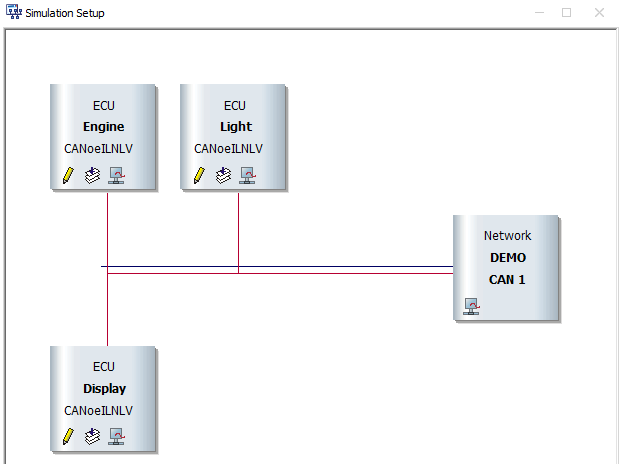


Figure 8. Simulated CAN Network.

# Design

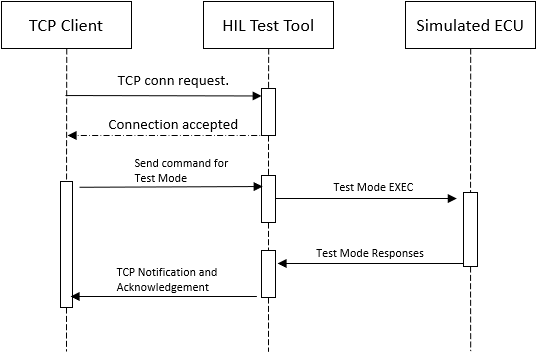
This section describes the design approach used for the project. Figure 9 shows the Flow Diagram for a single test mode command. The project has 3 sections: Hardware, Software and Test. Figure 10 shows the flow diagram for each area of the project.

Figure 9. Flow Diagram for a single test mode command.

## Project Development Process

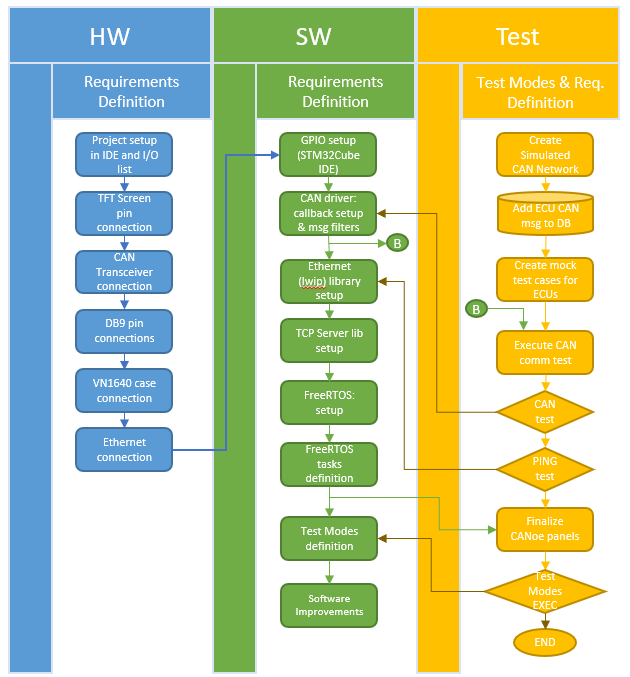


Figure 10. Flow Diagram for Task development.

The system development process that was chosen was the waterfall model. For this project, the requirements definition was static, all requirements for the 3 sections were defined in the project proposal and approved so there was no need to have requirement modification or reviews while the project was in progress.

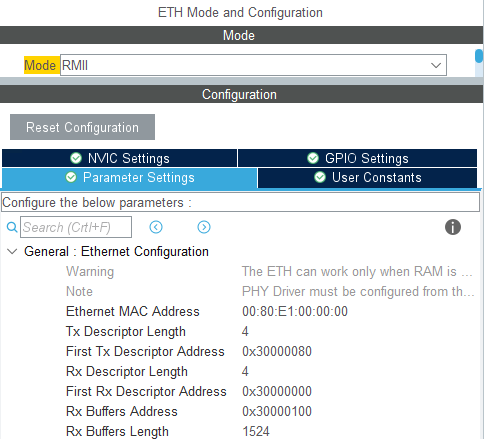
Once the Hardware was setup and all devices connected the next phase was the software design, in this phase as seen in Figure 8 the CAN driver, Ethernet, TCP Server and FreeRTOS were configured and integrated in the HIL Test Tool side, on the other hand the CANoe configuration was defined and panels, variables and CAN communication was enabled to receive the messages from the HIL Test Tool.

For the Integration and System Testing phase both parts of the project were put together and verified that the HIL Test Tool was sending the current CAN messages and 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.

| SDP Phase | System Development Process (SDP) | |
| --- | --- | --- |
| Section | Activity |
| Req. Definition | Hardware (HW) | Component list, devices, and connection definition. |
| Req. Definition | Software (SW) | Software and task requirements. |
| Req. Definition | Test | Test Mode definition and coverage. |
| SYS and SW design | Hardware (HW) | Project setup in IDE and I/O definition. |
| SYS and SW design | Hardware (HW) | TFT Screen pin connection. |
| SYS and SW design | Hardware (HW) | CAN Transceiver connection. |
| SYS and SW design | Hardware (HW) | DB9 pin connection. |
| SYS and SW design | Hardware (HW) | VN1640 case connection. |
| SYS and SW design | Hardware (HW) | Ethernet connection. |
| SYS and SW design | Software (SW) | GPIO setup and const name creation in IDE. |
| SYS and SW design | Software (SW) | CAN driver configuration, callback setup and message filters creation. |
| SYS and SW design | Test | Create simulated CAN network in CANoe. |
| SYS and SW design | Test | Create and add test CAN msgs to database in CANoe. |
| SYS and SW design | Test | Create mock test cases for ECUs. |
| SYS and SW design | Software (SW) | Configure Ethernet communication in IDE and generate code. |
| SYS and SW design | Software (SW) | Integrate TCP Server library. |
| Integration and Testing | Test | Perform basic IP address from TCP client to HIL Test Tool. PING test. |
| SYS and SW design | Software (SW) | Setup FreeRTOS and task creation (Application Code). |
| Integration and Testing | Software (SW) | Created Test Mode code and |
| Integration and Testing | Test | Executed Test Modes |
| Operation & Maintenence | Software (SW) | Software improvements |
| Operation & Maintenence | Test | ECU communication improvements |

Table 6. Summary of System Development Process.

### Controller Area Network (CAN)

The Nucleo-H723ZG has 1 CAN controller with FD support. This project only uses the controller as CAN High Speed. Figure 10 shows the most important parameters to achieve 500 kb speed with an input clock speed of 40Mhz:

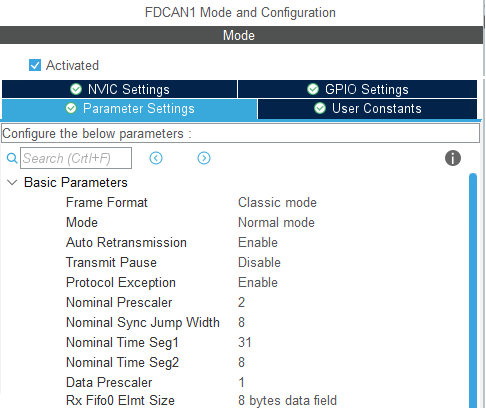


Figure 11. CAN basic configuration.

Figure 11 shows the calling graph of the CAN feature. The left side starting with MX\_FD\_CAN1\_Init shows to how the driver is being initialized with the parameters shown in Figure 10. 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.

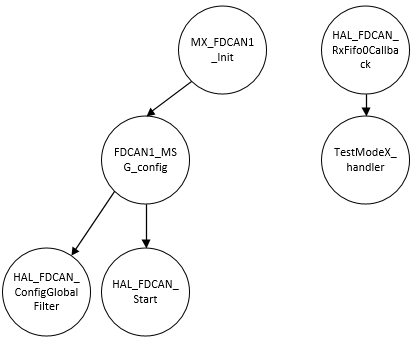


Figure 12. CAN communication Calling Graph.

### Ethernet/TCP Server

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

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

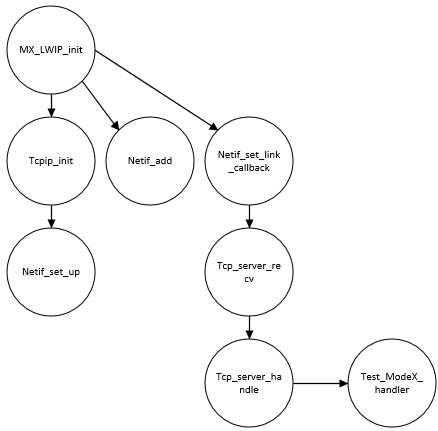
Figure 13. Parameter settings for MCU Ethernet address

Figure 14. Ethernet and TCP Server Calling Graph.

The middleware TCP/IP stack is the 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, configure the ports and IP address and start sending commands to the simulated ECUs via CAN.

### GPIO

Table 7 show the IO pins used for the project. Group A are the pins dedicated to connecting the TFT screen via SPI and Group B are the ones needed for Tx and Rx for the CAN transceiver.

| PIN | GPIO | | |
| --- | --- | --- | --- |
| Name | Device | Description |
| PA5 | SPI1\_SCK | TFT screen | SPI clock |
| PD7 | SPI1\_MOSI | TFT screen | Main Output Sub node In |
| PD0 | FDCAN1\_RX | CAN transceiver | Rx line for MCU |
| PD1 | FDCAN1\_TX | CAN transceiver | Tx line from MCU |

Table 7. GPIO list

### Real Time Operative System

The chosen OS was FreeRTOS. FreeRTOS provides a lightweight RTOS and allows modularization of tasks making it a good framework for this project.

### Appliction Code (Tasks)

A total of 7 RTOS tasks were created. Controller\_handler, . All the tasks are in freertos.c

#### Controller Handler

This is the main task of the application code,

### Interruptions

## Test Modes

# Test Results

# CONCLUSIONS

##### References

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*
2. J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.
3. I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.
4. K. Elissa, “Title of paper if known,” unpublished.
5. R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.
6. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**