2022 NDIA MICHIGAN CHAPTER
GROUND VEHICLE SYSTEMS ENGINEERING
AND TECHNOLOGY SYMPOSIUM
MODELING SIMULATION AND SOFTWARE (MS2) TECHNICAL SESSION
AUGUST 16-18, 2022 - NOVI, MICHIGAN

MULTI-LEVEL HARDWARE-IN-THE-LOOP TEST API FOR HARDWARE-SOFTWARE INTEGRATION TESTING

Michael Lingg, PhD[1], Timothy J Kushnier[2], Rodolfo Proenza, MS[2], Howard Paul[1], Brendan Grimes[1], Emory Thompson[1]

[1]Array of Engineers, Grand Rapids, MI
[2]DEVCOM GVSC SEC, Warren, MI

ABSTRACT

Hardware/software integrated system ensures a system will operate as intended in the same configuration it will be used in the field. Manual system testing can be a very slow and error prone process, as well as being incapable of testing interfaces that humans cannot interact with. Many existing solutions exist to introduce test hardware into the loop for verifying systems, but most of these solutions provide a separate component for each hardware interface. This paper presents an approach for a single integrated system that can test all hardware interfaces of a system under test, managed by a single controller. This test system provides the capability to abstract away the hardware being tested so a test developer can develop tests while only understanding the manual interfaces of the system being tested. We show that this approach can provide a significant acceleration to the time to execute tests, as well as improving the reliability, and consistency of the tests.

Citation: M. Lingg, T. Kushnier, R. Proenza, H. Paul, B. Grimes, E. Thompson, "Multi-level Hardware-In-The-Loop Test API For Hardware-Software Integration Testing," In *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium* (GVSETS), NDIA, Novi, MI, Aug. 16-18, 2022.

1. INTRODUCTION

Integrated hardware/software systems are invaluable in modern equipment. While pure hardware systems often require complete replacement to expand or add new capabilities, integrated hardware/software systems can adapt to new features being added to the system with updated software. This update capability provides

two primary benefits. First the ease of adding new features can extend the life of the equipment, which may otherwise become obsolete as technology advances. Second, software updates can allow a system to adapt to a rapidly changing environment. Over the air software updates allow equipment to be repurposed, or provided extended capabilities, in the field [1].

The flexibility provided by the integrated hardware/software system can lead to increased complexity when testing the system. During initial development, software unit testing may be performed on a PC without the actual hardware, and some hardware functionality can be tested with a basic test stand. However these separate test methods often fail to find bugs that show up when the software and hardware are integrated together. Integration testing of an integrated hardware/software system can be performed by a human working with the interface and observing the outputs of the system, but these systems increasingly have multiple components communicating with each other. In most cases, the interactions between multiple components use hardware communication protocols that require specific hardware to interface with, delaying testing until all systems are available to work together, or requiring a hardware test interface. Automated hardware testing can allow for a greater number of tests to be executed in a shorter period of time, allowing for a greater number of boundary conditions to be tested, or simply reducing the amount of time required for testing.

Automated hardware in the loop testing can allow for earlier testing during development. As soon as hardware is available, the hardware in the loop tests can be ready to run as the software implementation becomes more complete. This earlier testing allows the developers to receive feedback on their implementation well before the end of development. As development continues, the test suite can continue to be used for regression analysis, to ensure that while new features are implemented correctly, existing functionality is not broken. The same hardware in the loop test system can also be adapted to be used on the production line, ensuring all functionality works as expected on each unit as it is produced, or to diagnose hardware issues in the field. Finally the automated hardware in the loop testing can be used to test new features on an identical hardware software integration system back at the shop, before deploying to the field. The updates can be quickly tested for regression to ensure all features work as expected, providing high confidence that deploying the software will have the desired effect.

The focus of this paper is to provide a hardware in the loop test approach that reduces the time necessary to fully test an integrated hardware/software system, with a particular focus on reducing the time, and technical knowledge required, to develop tests. To provide a proper framework we will next look at three examples for testing components of an integrated hardware software system, and how test developers and the rest of the system would interact with these components. Then we will look at existing hardware in the loop testing solutions. Finally we will present a new all in one hardware in the loop test approach. Then look at test performance results showing the benefits of this system in time to execute a suite of tests, and reducing weeks long manual tests to days. Finally we will discuss the consistency provided by the test interface, and automation of test execution.

2. BACKGROUND

First we will look at some possible components of an integrated hardware/software system to be tested. In these examples we consider how these components work in a complete system, with no test hardware connected to the system under test. After defining how each component interacts with the system controller, an example system with all components will be described, including manual test examples.

2.1. Simulating a Multi-Function Display

The first component looked at is a multi-function display (MFD). In the complete system, inputs from the MFD would lead to the system activating other components of the system, or displaying data from other components of the system, for example vehicle climate control. The MFD communicates with the system controller via a Controller Area Network (CAN) bus [2]. The system controller will send a periodic Heartbeat message that includes the current screen the MFD is expected to display, and the system will go into a fault mode if the MFD does not acknowledge the heartbeat. Any button press by a human tester on the MFD is sent as a single message from the MFD over the CAN bus. The button press message will include the identifier of the button pressed.

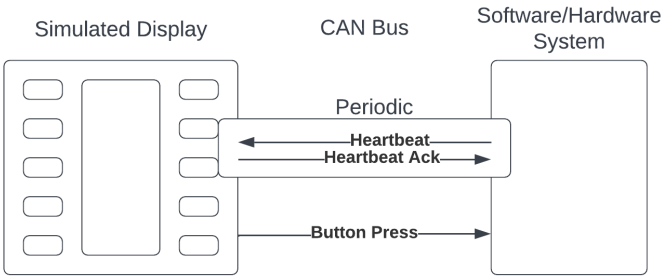

Figure 1: Example Multi-Function Display.

A basic test for this example would start with powering up the system with the MFD attached, then verify after a time limit that a fault is not displayed on the screen. Next a button would be pressed by a human tester, then the human tester will verify the display changes to the appropriate screen within a time limit.

2.2. Testing control of a speaker

The second component for the system is a speaker. This component is rather simple on the surface, the speaker is commanded by the system to output a given frequency at a given volume. One possible manual test is extremely simple, if rather imprecise. Have a human tester listen to the speaker and decide if it sounds right. Another possible option is testing the speaker sound output with a microphone and analyzing the signal. The microphone would require a fairly controlled environment if a precise measurement of the sound output is desired.

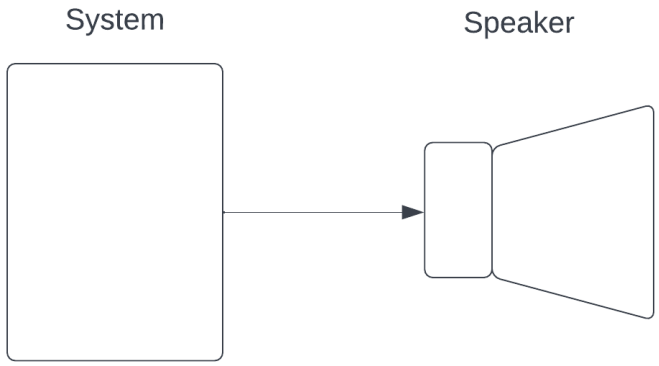

Figure 2: Speaker Test.

2.3. Testing reading of a thermocouple

The final component is a thermocouple that outputs a voltage to be translated by the system into a temperature. There are limited simple test methods to test this with any accuracy. A thermometer can be placed next to the thermocouple by a human tester to measure the present temperature, and a fan, or heater, could be used to decrease or increase the temperature, but the thermocouple and thermometer may change temperature at different rates. An expensive temperature chamber can test the system with high precision, but the size of the system under test may limit practical use of a temperature chamber.

2.4. Combined system

With these components we can define how the system works as a whole to act as a high temperature alarm. The temperature sensor provides the current temperature to the system, and the speaker sounds an alarm if a high temperature condition exists. The MFD provides an alarm menu system to set the temperature threshold for the alarm condition.

A simple manual test procedure might be for a human tester to power up the system with all hardware connected, verify the main menu is displayed on the MFD, not a Fault menu. Then the human tester would press the button to enter the alarm menu. Next the human tester would press the increase or decrease temperature button until the temperature threshold is one unit below the current temperature, and verify the alarm sounds. Finally the human tester would press the increase temperature button twice, and verify the alarm does not sound.

2.5. Existing solutions

While human testing of systems often has low up front costs, the time required for human testing can quickly add up, and often humans are simply not equipped to properly test many digital and analog systems. For example a human can verify the menu switching for the MFD, but if the MFD is disconnected a human cannot determine if, or when, the system entered fault mode. Simple hardware solutions can be tailored to specific tests. An oscilloscope can be connected to the CAN bus to record screen update messages, but then the human tester will need to decode the CAN signals and many oscilloscopes have a limited capture window. A better solution for this test application is to replace the MFD with simulation hardware that can read and respond with digital signals at very high speed, and includes the capability to process and decode the data. This approach allows button presses to be sent by a script, rather than physically pressed by a human tester. While it may be possible to inject simulated button presses with the MFD attached, the MFD is being replaced with simulated hardware in this example, as it is assumed that the MFD CAN traffic would conflict with simulated CAN traffic.

National Instruments (NI) provides an excellent range of hardware test products [3]. Among other products is a CAN analyzer [4] that could work nicely for simulating the MFD in this example. Total Phase also produces a similar analyzer [5]. NI also provides the LabVIEW [6] program that allows their hardware products to be programmed for specific operation, and to analyze data read by their hardware.

As powerful as these existing products are, they tend to focus on the signals that the NI product is designed to work with. To use these devices, a test developer will need to understand the hardware signals involved. For our MFD example, understanding what the system should do from a user perspective is not sufficient, the test developer will also need to understand what CAN messages are being communicated, and sometimes details of how the CAN protocol works. Additionally while some of these existing hardware test products can be connected together to simulate, and/or read, multiple signals in parallel, the products are typically sold separately and require the test developer to determine how to connect the devices and program them to work together.

An alternative is creating an integrated system that manages all necessary hardware interfaces in a single unit. Benefiting this approach, costs of producing circuit boards have decreased significantly. Today a circuit board capable of connecting to all sensors and communications ports of a small to medium system can be printed, even in small quantities, for a unit cost equivalent to tens of man hours of a human tester's time. This makes integrating all of the necessary elements to test a system into a single board, that includes its own processing unit, very cost effective. With all of the testing components integrated together, a test scripting interface can be developed that abstracts away the low level details of the components being tested. This abstraction means tests can be developed that only requires high level knowledge of how the system works, no longer requiring test developers to understand the protocols involved. Next the details of such a system will be discussed.

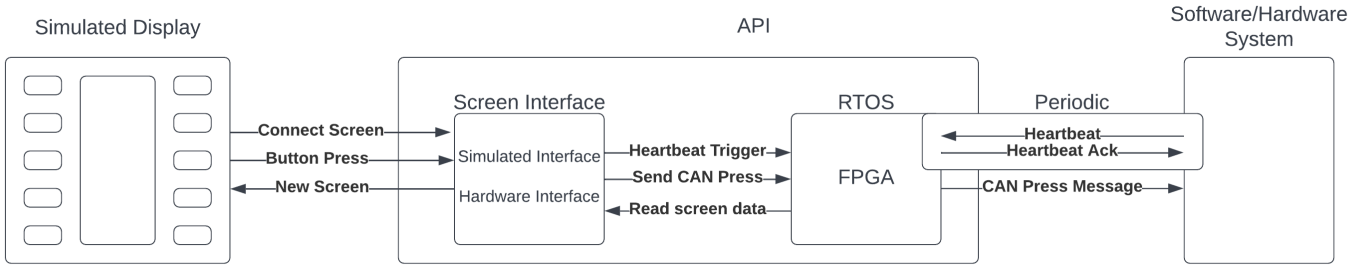

Figure 3: Multi-level API.

3. METHODS

In this section a Multi-level Hardware-In-the-Loop (MHIL) test system will be described, from here on referred to as the MHIL test system. The discussion will start with the hardware protocols from the system under test example provided in the Background section, and the abstraction layers will be described at higher and higher levels, until a simple interface is reached that mimics a system under test. This is illustrated in figure 3.

3.1. Hardware

To satisfy the system under test example above, the MHIL test system would include at a minimum a processor, an FPGA, a CAN transceiver, an Analog to Digital Converter (ADC) for analyzing analog waveforms, and a Digital to Analog Converter (DAC) for outputting commanded voltages and analog waveforms. These components are assembled on a single circuit board to provide maximum integration. Integrating the processor and FPGAs into a single package, System on Modules (SoM) [7] are fast becoming a standard tool for advanced embedded processing needs as they provide the powerful benefit of a simple high-performance memory mapped interface between processor and FPGA. The interface allows for massive amounts of data to be captured by the various modules on the FPGA, and for this data to be directly written to system RAM, ready for access by the processor. The MHIL test system as described in the next subsections also includes an Ethernet connection for communication with a host PC, providing additional flexibility. The circuit board also provides a simple header connection for all input and output ports in the MHIL test system, and custom cables can be constructed to connect the simple header to the system under test connectors, ranging from custom style header connections, to ADC test points, to standard or MIL spec connectors.

As discussed in the previous section, the MHIL system will replace interfaces humans use for interaction with the hardware described above. The MFD can be replaced with a CAN bus connected to the MHIL system that can send simulated button presses, and log screen updates the system under test sends over CAN. The speaker can be replaced with an ADC to record waveforms being sent to the speaker. The thermocouple can be replaced with a DAC to send voltages representing the thermocouple output for simulated temperatures.

3.2. FPGA

The FPGA(s) is used to directly control the hardware elements on the MHIL test system. A high speed processor with a Real Time Operating System (RTOS) can sample high speed data with a consistent sample rate. However an FPGA can provide better timing guarantees, and an FPGA can manage multiple ADCs with high sample rates, while

managing multiple DACs, CAN ports and other hardware interfaces, all in parallel. Additionally the FPGA can be used as a common clock between all of the hardware the FPGA manages, ensuring data is sent at the desired time, and received data is timestamped, with very high precision. An FPGA running on a 100 MHz clock can ensure all outputs are sent, and all inputs are timestamped, with a 10 nanosecond precision, as long as they are all running through the FPGA.

For controlling the DAC, the FPGA converts desired DAC voltage values DAC counts, as well as outputting samples from a repeating waveform, if desired. For the ADC, the FPGA reads and stores the ADC samples in RAM for access by the RTOS. For the CAN bus the FPGA manages the digital bits of the CAN protocol for sending and receiving messages. At all times the FPGA monitors the CAN port to handle arbitration (ensuring the managed port does not send messages that collide with other messages active on the bus), and to process CAN messages received from other nodes.

3.3. RTOS

The basic function of the RTOS is to manage when the FPGA is commanded to update hardware outputs, logging data received to alleviate limited FPGA memory, and sending hardware configurations to the FPGA. Configurations might include elements such as BAUD rate, or sample rates, and configurations can include device specific calibration values for the components, such as the DAC or ADC. An RTOS is used to ensure messages are sent at precisely the desired time. This interface provides some abstraction of the values the hardware sends out, such as producing the PWM (Pulse-Width Modulation for increasing the precision of the DAC voltage output) waveform necessary for a commanded DAC voltage, or generating the DAC output samples, such as for a sine wave, when provided the desired waveform, frequency and amplitude. Similarly a set of CAN messages can be sent to the RTOS from the host PC, with times to send the messages, and the RTOS ensures the timing of the commanded messages.

The real value of the RTOS is in managing periodic messages, and request response messages. The RTOS can be commanded to repeat a given output, similar to the CAN heartbeat message, and the RTOS will keep sending the message at the commanded rate. The RTOS will also manage automatic updates to the message, such as incrementing a count of how many heartbeats have been sent. The updates can also be used to modify the message being sent based on other data received from the system under test, an example of this could be acknowledging the requested MFD state in a CAN message. In addition to the periodic message, the RTOS can send messages based on trigger conditions. These trigger conditions can range from sending a single CAN message in response to a received CAN heartbeat, to updating a DAC output based on a value read by the ADC, or any combination of outputs, based on a combination of input conditions.

3.4. Host PC

In the current MHIL system, a host PC is used for execution of the test or simulation scenarios. The host PC software is broken into two levels. The lower level is a communication protocol that specifies how the MHIL hardware communicates with the system under test. The higher level is a simulation interface that provides functionality mimicking the system under test interface, translating this interface to the actual hardware in the system. More details of each subsystem is provided next.

3.5. Host PC Hardware interface

The hardware interface translates between the hardware elements in the system under test, and how the hardware elements are organized within the MHIL test system. In our example system under test the MFD interfaces with a CAN bus. In a more

complex system this may be one of multiple CAN busses. The hardware interface translates from the MFD CAN bus defined by the system under test, to how the MHIL test system refers to the same port, possibly simply CAN bus 1. This is extended further with the ADC system. The system under test may have a large number of analog test points that do not all need to be sampled in parallel. To accommodate this, the MHIL test system can use a multiplexer (MUX) to switch one or more ADCs between a greater number of test points. So when a system test requests the speaker voltage waveform be sampled, the hardware interface can set the appropriate MUX settings to sample the speaker voltage waveform at the desired time.

3.6. Host PC Simulation interface

Finally the simulation interface provides script functions that replicate the system under test. For our system under test example, we will focus on simulating the elements going into the system under test. Methods to test the MFD, thermocouple and speaker themselves are not described here. To accomplish this, the MHIL test system will simulate the MFD on the CAN bus, simulate the thermocouple voltage value via the DAC, and reading the voltage going to the speaker using the ADC.

The following functions would be provided for simulating the MFD:

- connect_mfd: This function would create a trigger condition so the RTOS will send an acknowledgement of received heartbeat message. The response will include the message identifier from the heartbeat message that triggered the response.
- disconnect_mfd: This function would remove the trigger condition to respond to the heartbeat.
- read_mfd_state: This function will return the commanded MFD state from the last received heartbeat message, including if a fault condition exists.
- push_button: This function will send the CAN button press message with one of the following button indexes:
  - alarm_menu_button: Enters the alarm menu, if the menu state is in the main menu.
  - temp_up_button: Increments the alarm temperature, if the menu state is in the alarm menu.
  - temp_down_button: Decrements the alarm temperature, if the menu state is in the alarm menu.

The following function would be provided for simulating the temperature sensor:

- set_temperature(in temperature): This function would translate from the desired temperature to the desired DAC voltage in degrees Celsius. The DAC voltage would be sent to the RTOS to be converted into a PWM waveform, which would be sent to the FPGA to set the DAC output.

The following function would be provided for reading the speaker:

- read_speaker(): This function would command the hardware interface to MUX an ADC to the speaker voltage test point, then command the RTOS to read the ADC waveform, as sampled by the FPGA, and return it to the host PC, to post process and find the current frequency and volume being sent to the speaker. Frequency and volume are provided as elements of this call.

3.7. Test example

A simple manual test example was provided in the background section. As discussed, some parts of the test, such as the exact timing of screen updates, and if a fault condition was set while the MFD was disconnected, cannot be tested manually. Now we will look at how this test might be performed with the MHIL test system with the example test script below.

Assumption: Default temperature alarm threshold is 50 degrees Celsius.

```
connect_mfd()

wait(test_delay)

current_state = read_mfd_state()
if current_state is not MAIN_MENU:
   report_error()

push_button(alarm_menu_button)

wait(test_delay)

current_state =
   read_mfd_state().get_menu_state()
if current_state is not ALARM_MENU:
   report_error()

set_temperature(51)

wait(test_delay)

speaker_signal = read_speaker()
if speaker_signal.frequency
  is not ON_FREQUENCY:
   report_error()
if speaker_signal.volume
  is not ON_VOLUME:
   report_error()

set_temperature(49)

wait(test_delay)

speaker_signal = read_speaker()
if speaker_signal.frequency
  is not OFF_FREQUENCY:
   report_error()
if speaker_signal.volume
  is not OFF_VOLUME:
   report_error()

disconnect_mfd()

wait(test_delay)

current_state =
   read_mfd_state().get_menu_state()
if current_state is not FAULT_STATE:
   report_error()
```

This test can cover everything required in the manual procedure, while the script is easily readable by someone who understands the system under test, but may not understand CAN busses, or DACs. Beyond being easy to write, the test includes very precise timing in that it can check the state of the system after precise, computer controlled delays, to ensure the desired behavior happens at the required time.

4. RESULTS

The MHIL system provided above is a simple example for discussing the capability of the system, potential systems are limited only by the number of SoMs used in the system. One system we have been using for testing integrated hardware/software systems includes the following components:

- 1 UART port.
- 4 UART ports
- 2 CAN networks
- 4 DACs
- 6 ADC channels
- 42 GPIOs
- 1 Ethernet port

Note: UART ports can be configured as RS232 or RS485. DACs range from 0 to 10V, accurate to 10 mV before dithering. ADC channels are MUXed to 16 analog input ports. 2 ADC channels capable of 25 (Million samples per second) MS/s max capture rate and 4 ADC channels capable of 1MS/s capture rate.

In addition, the host PC is capable of interacting with a large variety of other interfaces, for example:

- 1 iBoot
- 1 Eithernet interface
- 1 USB

Note: Ethernet is used for Data Distribution Service (DDS) communication.

This configuration is intentionally over engineered for any single system to be tested. Combining extra capability into a single MHIL system provides minimal increase of cost (10-20%), while allowing us to use the same MHIL system to test system under test configurations, without needing to spin up a new MHIL system for each system under test. Having this many hardware interfaces integrated into a single MHIL system allows significant parallel capability of the interfaces. The MHIL system can send separate messages on each of the two RS232 ports and each of the two CAN networks, as well as toggling GPIOs and capturing on all six ADC channels, with millisecond accuracy of commanded updates from the RTOS. For capturing timing of outputs and inputs, the FPGA can be used to synchronize the system.

One system under test includes multiple tests that use various combinations of 1 UART port, and 1 RS232 port, both CAN ports, 3 DACs, 14 of the analog input ports, and 5 GPIO pins. Testing the system includes 110 tests to execute, these executions consist of 70 base tests, of which some were required to be executed on multiple units. Between test execution time, time to document results, and general overhead time involving a manual tester, each person could execute on average 5 tests per day. This resulted in 22 work days to execute the tests, or 4.4 weeks under ideal situations. Often these tests had to be executed multiple times due to user error. Further each failure results in time to analyze the reason for the failure and ensure the next execution will succeed. This failure rate and time for analyzing failures added an additional 50% to the overall execution time, extending time to completely test the system to approximately 6.6 weeks. The MHIL system produces consistent test results and can be run 24/7. On average each test can be executed by the MHIL system in 20 minutes. This means the MHIL system can run all 110 tests in 55 hours, or less than 3 days.

A second system analyzed includes DDS communications, 1 CAN port and 1 ADC port. This system has 10 tests to execute. Considering only the time to execute each test manually, this set of tests took 184 hours. The automated test can be run in just under 40 hours. With the current system under test, there are some steps that require some human intervention to execute the test. Of the 40 hours to execute this set of tests, just under 4 hours requires a human to be present. Note these test have limitations to how much automation can improve the execution times as the system under test includes timers the MHIL system has to wait on. The test execution time is further broken down as follows in figure 4.

The MHIL system also provides additional benefits beyond the simple acceleration of test execution. A script interface with functions that correspond with the customer's system under test can provide an easy to use interface, and reducing the training necessary for test developers to use the system. While reducing the necessary training of test developers, the test automation does also lead to the need for more precision in requirements and tests. While ambiguous requirements can be liberally interpreted by a human tester, often with passing results, the automated system requires precise values. This leads to increased cost of requirements, and sometimes of the tests themselves, but while the ambiguity can lower costs it also can allow a

number of errors to slip through testing. The ambiguity can also mean that a new tester interprets the requirements differently, and will fail the test. The automation of the test execution ensures tests perform the same operations, and run for the same duration, each time a test is executed, depending on any delays due to the system under test. Additionally, minor changes to requirements can be overlooked by testers so a manual test may not need to be rewritten, but minor changes will often lead to an update of the automated test. While this is also an increased cost, this can provide beneficial additional review, and validation that each change is correct.

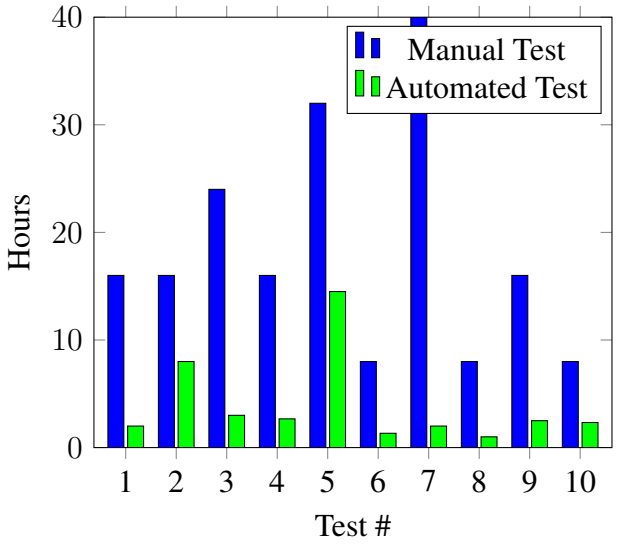

Figure 4: Second system test timing breakdown.

5. CONCLUSION

Reducing test time through test automation has been proven effective in reducing costs, and identifying bugs in the system. With the continued reduction in cost of producing circuit boards, providing a single integrated test system capable of performing hardware in the loop testing of a hardware software integrated system is not only practical, but also cost effective. We have provided a template for designing such an integrated system, as well as describing a high level test interface that abstracts away the specific hardware under test, and presents the test environment in a way that anyone familiar with the customer system will be comfortable with. We have provided timing results showing this system provides benefits similar to other automated systems in reducing time to execute tests, in one case reducing a manual set of tests requiring 6.6 weeks to execute, being run in 3 days with the MHIL system. We also discussed the less quantifiable benefits of how the system can make test development easier, more repeatable, and less ambiguous.

We look to extend the capabilities of this system further by seeing how the following additional features benefit the system:

- Adding RTOS monitoring to validate during run time how well tasks are executing at the proper rate, and using this information to provide warnings if the number of operations being performed have a chance of overrunning deadlines in a worst case scenario.
- Adding a model based IDE to extend the customer focused test scripts to a drag and drop environment, further simplifying test development with the system.
- Adding the capability to scan customer Interface Control Documents (ICDs) so that all messages can be presented to test developers with the required message components.
- A simplified lower cost unit that excludes the analog components for systems under test that only have digital signals.
- Networking multiple MHIL systems for testing multiple systems under test at the same time, or systems under test with a very high number of hardware connections.

6 References

[1] "Tesla over-the-air updates," theverge.com, March 2022. [Online]. Available: https://www.theverge.com/2018/6/2/17413732/tesla-over-the-air-software-updates-brakes

[2] "Can bus," wikipedia.com, March 2022. [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus

[3] "Ni hardware," ni.com, March 2022. [Online]. Available: https://www.ni.com/en-us/shop.html#pinned-nav-section2

[4] "Ni can interface device," ni.com, March 2022. [Online]. Available: https://www.ni.com/en-us/shop/hardware/products/can-interface-device.html

[5] "Komodo can duo interface," totalphase.com, March 2022. [Online]. Available: https://www.totalphase.com/products/komodo-canduo/

[6] "Labview," ni.com, March 2022. [Online]. Available: https://www.ni.com/en-us/shop/labview.html

[7] "System on module," wikipedia.com, March 2022. [Online]. Available: https://en.wikipedia.org/wiki/System_on_module