Development of a Hybrid CAN Testbed for In-vehicle Security Research

William Luke Lambert, Dept. of Computer Science, Tennessee Tech University, Haley Burnell, Dept. of Computer Science, Tennessee Tech University, and Dr. Sheikh Ghafoor, Dept. of Computer Science, Tennessee Tech University

Introduction

- A modern automotive system consists of up to 100 Electronic Control Units (ECUs) and thousands of communication signals via the Controller Area Network (CAN) protocol, over a bus-based network topology [3,4].
- While the CAN protocol benefits from its robust low-cost, reliable, and real-time properties, it lacks information security mechanisms, such as authentication and encryption [1].
- In-vehicle security researchers today use 2 main approaches to study CAN: simulated ECUs and actual vehicle ECUs.
- While simulated ECUs are often cost-effective and provide greater flexibility, they do not typically provide realistic results.
- On the other hand, actual ECUs provide a realistic scenario, however, they can be cost-prohibitive and lack flexibility when modifications need to be made.
- Our main research objective is to design and develop a hybrid CAN testbed that incorporates both real and simulated ECUs, to provide both the real time behavior of actual ECUs and the flexibility of simulated ECUs.
- We have also created an easy-to-use software interface to aide in software development on the testbed.
- Our testbed has been validated with a functionality test and a secure CAN protocol, SecCAN [2], has been implemented.

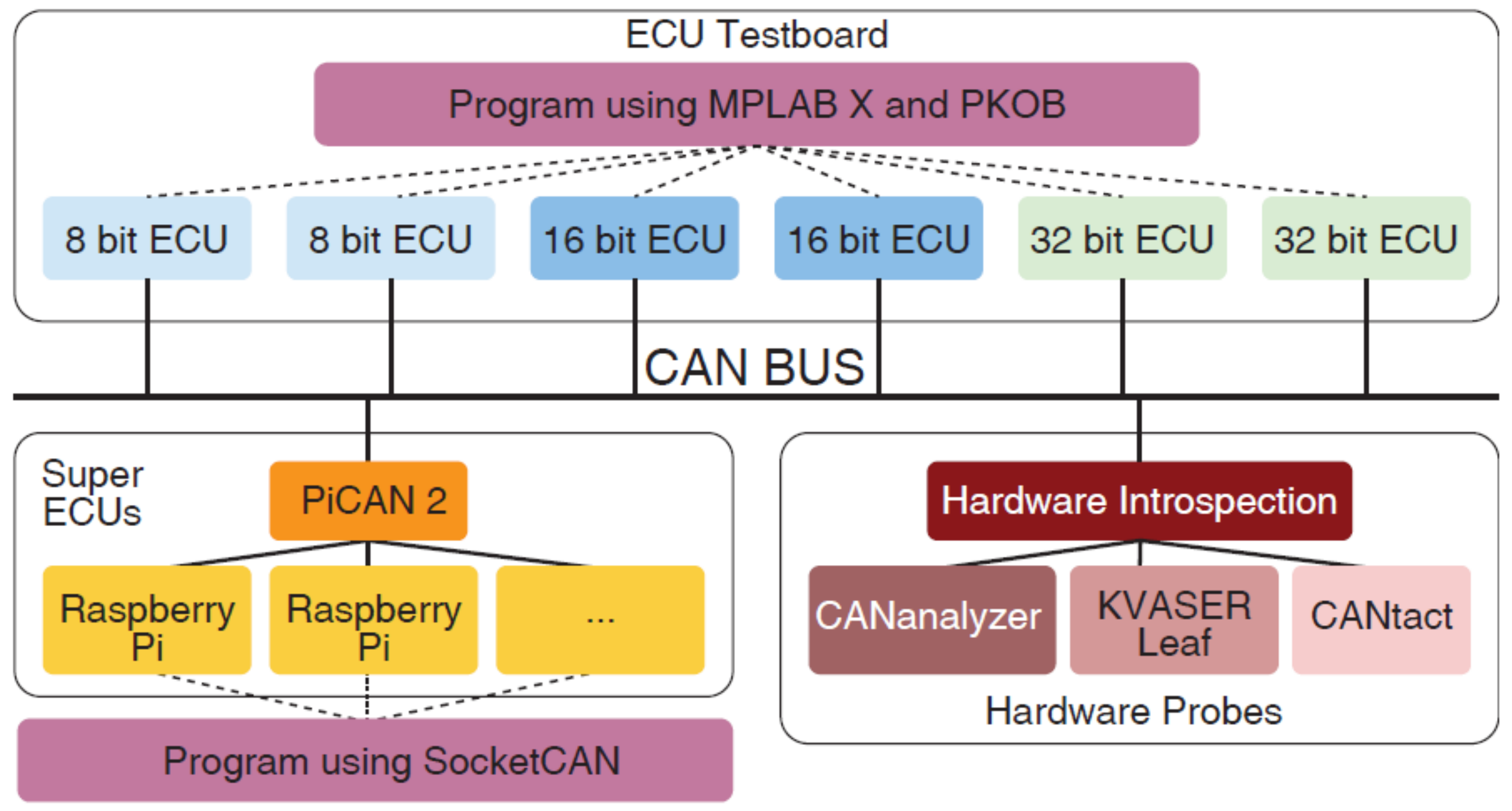


Fig. 1. Preliminary High Level Design of Hybrid CAN Testbed

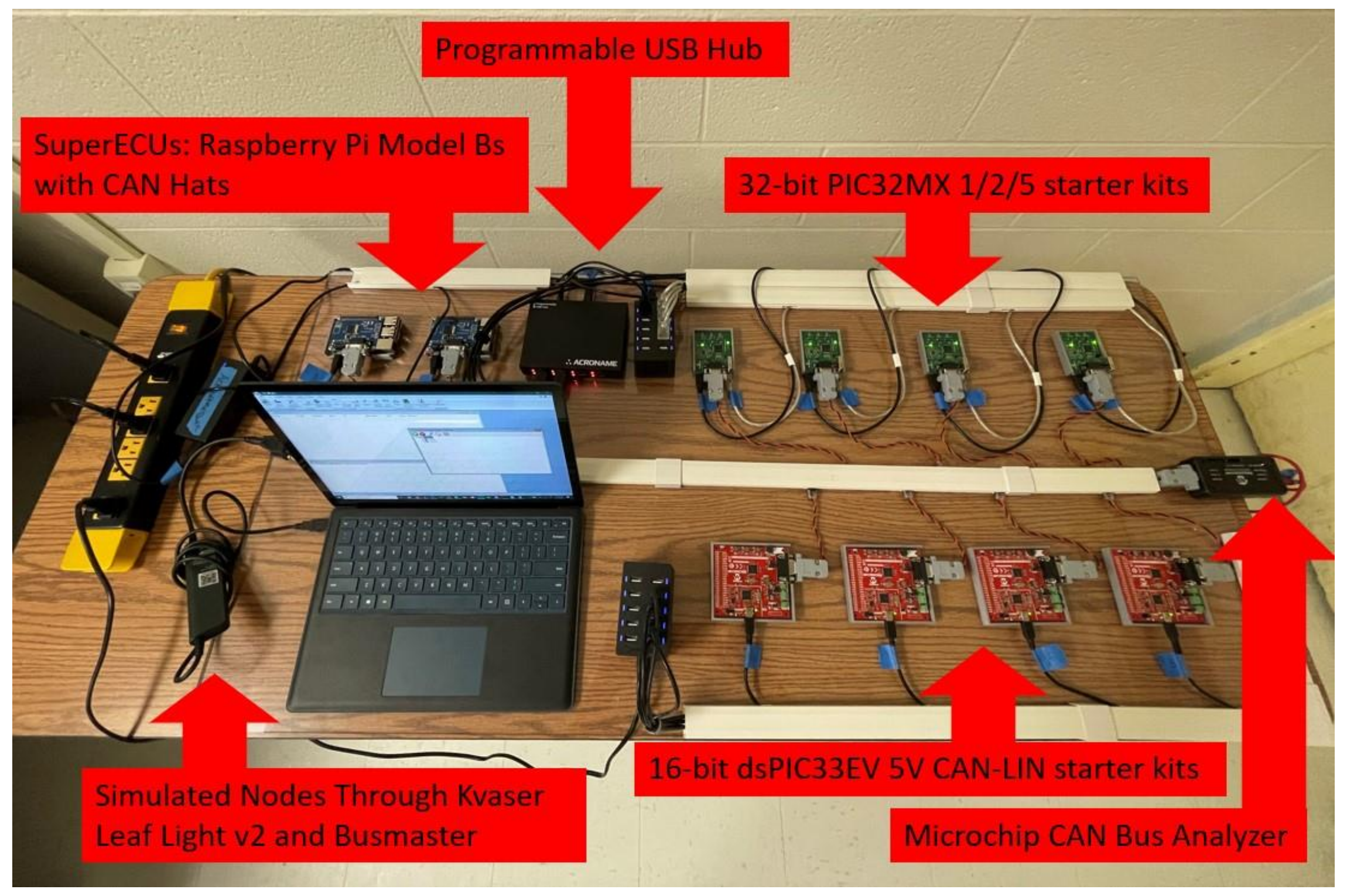

Programmable USB Hub
SuperECUs: Raspberry Pi Model Bs with CAN Hats
32-bit PIC32MX 1/2/5 starter kits
Simulated Nodes Through Kvaser Leaf Light v2 and Busmaster
16-bit dsPIC33EV 5V CAN-LIN starter kits
Microchip CAN Bus Analyzer

Fig. 2. Hybrid CAN Testbed

Design and Development of the Hybrid CAN Testbed

Hardware

The hardware side of our testbed is constructed based on the design in Figure 1, with the following main components, seen in Figure 2:

- **ECU Test Boards:** These are used to emulate the actual ECUs in the vehicle. We have used the Microchip 16-bit dsPIC33EV 5V CAN-LIN and 32-bit PIC32MX 1/2/5 starter kits.
- **Simulated ECUs:** These are software-based ECUs created using the Busmaster simulator.
- **SuperECUs:** Raspberry Pi 3 Model Bs are used to simulate both current and future automotive ECUs. These are connected to PiCAN 2 Boards, which allow the Pis to interface with the CAN bus.
- **Probes:** The Kvaser Leaf Light v2, the Microchip CAN Bus Analyzer, and the PiCAN 2 Boards can all be used as hardware probes to sniff CAN traffic.

Software

We have also designed a software interface with the goal of combining, speeding up, automating, and enhancing the ECU software development, seen in Figure 3. So far, the following modules have been implemented:

- The **selection module,** seen in Figure 4, provides a way to select both source code needed to flash the ECU and a set of tasks useful for testing. The source code is compiled to create hexadecimal-format machine code (hex code), which is passed onto the flashing module.
- The **flashing module** takes both the generated hex code from the selection module and the selected ECU as input. An ACRONAME 8-Port Programmable USB HUB is used here to allow specific 32-bit ECUs to be flashed, even while the bus is running.

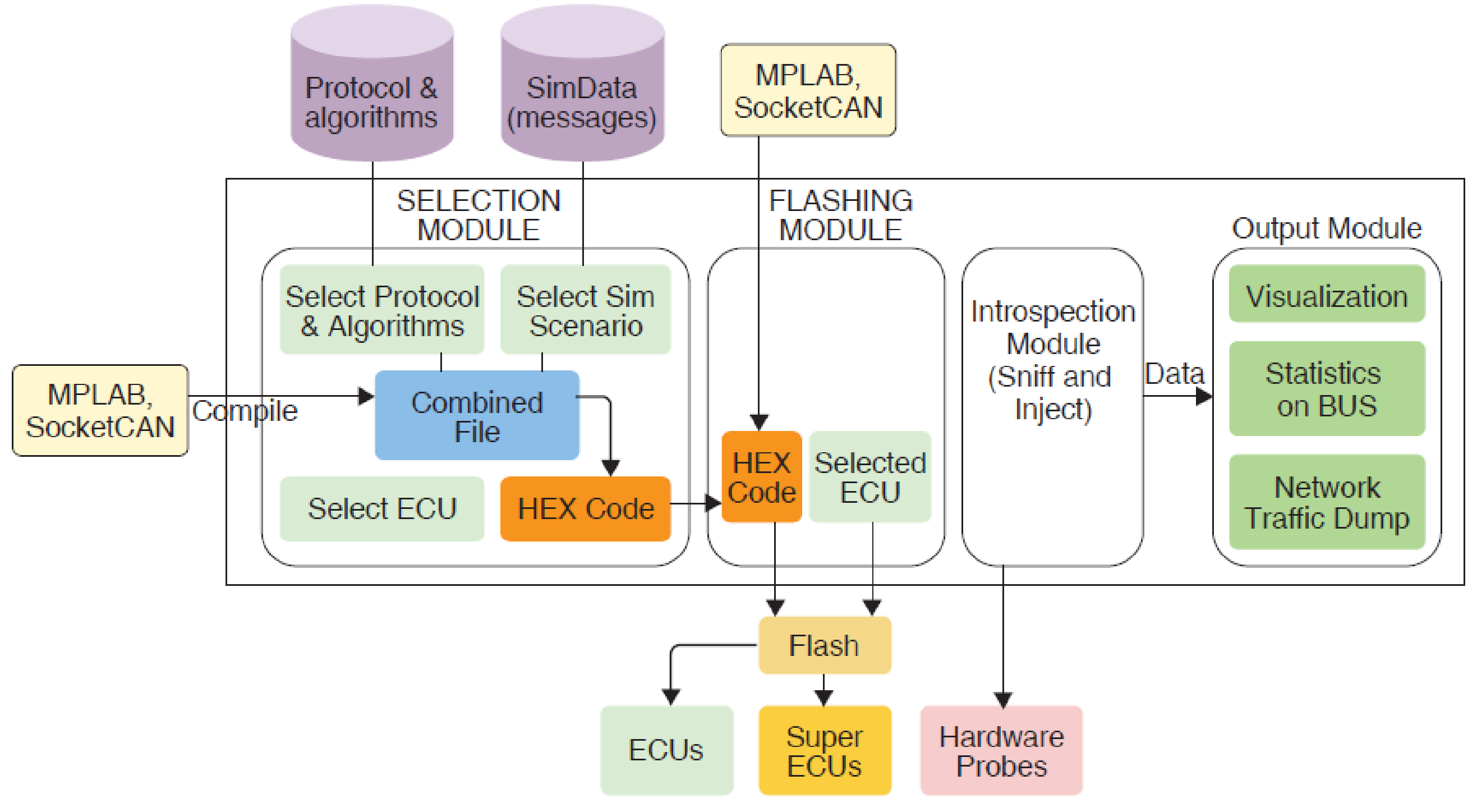
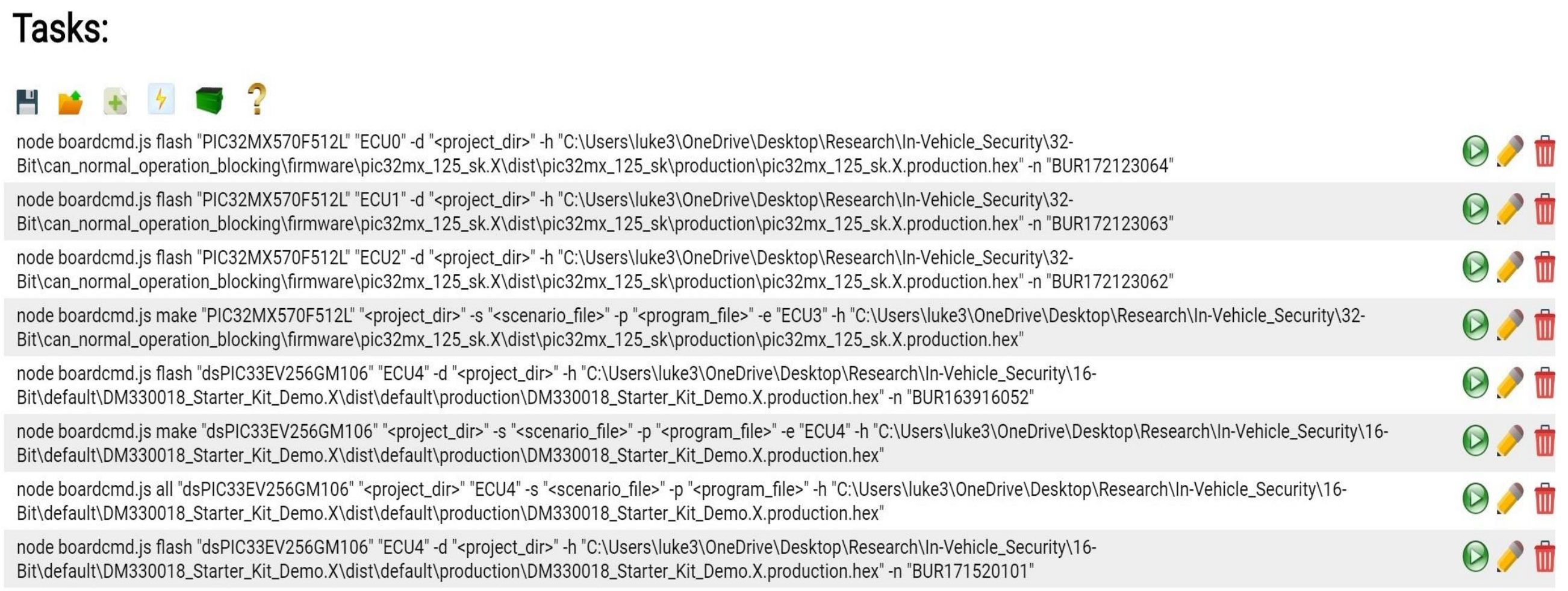


Fig. 3. High Level Software Design

The following modules are still in development:

- A **source code editing module** will work in tandem with the flashing module and will allow users to edit ECU source code on the fly before it is flashed to an ECU.
- A **hardware introspection module** will be used to sniff or inject messages into the CAN bus through one or more interface(s). This will be especially useful for analyzing real automotive events.
- An **output module** will take data from the hardware introspection module use various visualization techniques to represent the data


Tasks:

```
node boardcmd.js flash "PIC32MX570F512L" "ECU0" -d "<project_dir>" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\32-Bit\can_normal_operation_blocking\firmware\pic32mx_125_sk.X\dist\pic32mx_125_sk\production\pic32mx_125_sk.X.production.hex" -n "BUR172123064"
node boardcmd.js flash "PIC32MX570F512L" "ECU1" -d "<project_dir>" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\32-Bit\can_normal_operation_blocking\firmware\pic32mx_125_sk.X\dist\pic32mx_125_sk\production\pic32mx_125_sk.X.production.hex" -n "BUR172123063"
node boardcmd.js flash "PIC32MX570F512L" "ECU2" -d "<project_dir>" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\32-Bit\can_normal_operation_blocking\firmware\pic32mx_125_sk.X\dist\pic32mx_125_sk\production\pic32mx_125_sk.X.production.hex" -n "BUR172123062"
node boardcmd.js make "PIC32MX570F512L" "<project_dir>" -s "<scenario_file>" -p "<program_file>" -e "ECU3" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\32-Bit\can_normal_operation_blocking\firmware\pic32mx_125_sk.X\dist\pic32mx_125_sk\production\pic32mx_125_sk.X.production.hex"
node boardcmd.js flash "dsPIC33EV256GM106" "ECU4" -d "<project_dir>" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\16-Bit\default\DM330018_Starter_Kit_Demo.X\dist\default\production\DM330018_Starter_Kit_Demo.X.production.hex" -n "BUR163916052"
node boardcmd.js make "dsPIC33EV256GM106" "<project_dir>" -s "<scenario_file>" -p "<program_file>" -e "ECU4" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\16-Bit\default\DM330018_Starter_Kit_Demo.X\dist\default\production\DM330018_Starter_Kit_Demo.X.production.hex"
node boardcmd.js all "dsPIC33EV256GM106" "<project_dir>" "ECU4" -s "<scenario_file>" -p "<program_file>" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\16-Bit\default\DM330018_Starter_Kit_Demo.X\dist\default\production\DM330018_Starter_Kit_Demo.X.production.hex"
node boardcmd.js flash "dsPIC33EV256GM106" "ECU4" -d "<project_dir>" -h "C:\Users\luke3\OneDrive\Desktop\Research\In-Vehicle_Security\16-Bit\default\DM330018_Starter_Kit_Demo.X\dist\default\production\DM330018_Starter_Kit_Demo.X.production.hex" -n "BUR171520101"
```

Fig. 4. Selection Module

Functionality Test

- We have successfully tested the functionality of our testbed by ensuring communication is possible between all simulated, real, and super ECUs on the testbed
- To perform this test, a program was developed for each ECU, where ECUs in “send mode” will send a CAN message based on the combination of inputs (switches or keys) and ECUs in “receive mode” will receive these messages and light up corresponding LEDs.

Secure CAN Protocol (SecCAN)

- We have also implemented the novel secure CAN protocol, SecCAN [2], in a simulated environment using Busmaster and the 32-bit ECUs and preliminary results are promising.

Conclusion & Future Work

- In conclusion, this work contributes to the field of in-vehicle security by creating a hybrid CAN testbed to enhance the development and testing of future research in this field.
- In the future, we hope to develop a version 2 of our testbed with automotive ethernet. Specifically, 100Base-T1 (the IEEE's 802.3bw-2015) is being investigated. Automotive ethernet has the potential to enhance the speed of flashing the ECUs, in addition to providing additional bandwidth for machine learning applications, such as anomaly and intrusion detection systems.
- We also hope to develop a faster version of SecCAN, pending the analysis of additional hashing methods.

References

[1] De La Torre, G., Rad, P., & Choo, K. K. R. (2020). Driverless vehicle security: Challenges and future research opportunities. Future Generation Computer Systems, 108, 1092-1111.

[2] Mohammad, Arman Ullah. Sheikh Ghafoor, Stacy Prowell. SecCAN: A Practical Secure Control Area Network for Automobiles. To appear in the proceedings of 16th International Conference on Cyber Warfare and Security February 25-26, 2021, TN, USA.

[3] Umair, A., & Khan, M. G. (2018). Communication Technologies and Network Protocols of Automotive Systems. Advances in Networks, SciencePG, 6(1), 58-65.

[4] Voss, W. (2008). A comprehensible guide to controller area network. Copperhill Media.

Acknowledgements

I would like to thank the National Security Agency for funding this project and Dr. Stacy Prowell from Oak Ridge National Laboratory for his invaluable guidance. I would also like to thank my advisor, Dr. Sheikh Ghafoor, for the opportunity to pursue interesting research in the field of in-vehicle security, as well as Haley Burnell for her help with constructing our hybrid CAN testbed. Lastly, I would like to thank the author of [2], Arman Ullah, for his key help in the implementation and testing of his secure CAN protocol, SecCAN, on our testbed.