Ultimate Truck Hacking Platform

Unifying Truck Hacking Tools with the Power of Yocto Linux and BeagleBone Hardware

Dr. Jeremy Daily, Associate Professor of Systems Engineering
Rik Chatterjee, Graduate Student in Systems Engineering
Carson Green, Undergraduate in Electrical and Computer Engineering
What is the UTHP?

A Tool Revolutionizing Vehicle Diagnostics and Security:

- Unified Solution: Combining diverse truck hacking tools under one platform for seamless diagnostics and cybersecurity analysis.
- Yocto-based: Harnessing the flexibility of Yocto Linux to ensure adaptability and wide-ranging compatibility.
- Hardware Meets Software: Introducing a specialized BeagleBone based hardware designed to work hand-in-glove with our software build.
- Future-Proof: Regular kernel updates and platform porting to stay ahead in the fast-evolving world of vehicle technology.
Why do we need a new Platform?

- Enhanced Vehicle Security: As trucks become more technologically advanced, the need for comprehensive security diagnostics increases. UTHP offers an all-in-one solution to identify vulnerabilities and potential threats.
- Unified Toolset: Currently, professionals and enthusiasts must juggle various tools for different diagnostic tasks. UTHP offers a centralized platform, reducing the learning curve and increasing efficiency.
- Future of Transportation: With the rise of autonomous vehicles and smart transportation, it's crucial to stay ahead of potential cyber threats. UTHP provides a foundation for future-proofing vehicles.

To prevent something like this due to a cyberattack.
CyberTruck Challenge

Experience to guide in the design decisions and goals.
Participants

- Students
- Industry
- Instructors
- Mentors
Sponsors have access to some outstanding new talent in cybersecurity.
# CyberTruck Challenge 2023 Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>Sunday, 11 June</th>
<th>Monday, 12 June</th>
<th>Tuesday, 13 June</th>
<th>Wednesday, 14 June</th>
<th>Thursday, 15 June</th>
<th>Friday, 16 June</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group A</td>
<td>Group B</td>
<td>Group A</td>
<td>Group B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Before 0700</td>
<td>Site Closed</td>
<td>Site Closed</td>
<td>Site Closed</td>
<td>Site Closed</td>
<td>Site Closed</td>
<td>Site Closed</td>
</tr>
<tr>
<td>0700-0730</td>
<td></td>
<td></td>
<td></td>
<td>Breakfast</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0730-0800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0800-0830</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0830-0900</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0900-0930</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0930-1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000-1030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1030-1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100-1130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1130-1200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1200-1230</td>
<td>Lunch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1230-1300</td>
<td></td>
<td></td>
<td></td>
<td>Lunch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1300-1330</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1330-1400</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1400-1430</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1430-1500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500-1530</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1530-1600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1600-1630</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1630-1700</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1700-1730</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1730-1800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1800-1830</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Site Closed</td>
</tr>
<tr>
<td>1830-1900</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1900-1930</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1930-2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000-2030</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2030-2100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2100-2130</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2130-2200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>After 2200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **Site Closed:** All activities are closed.
- **Breakfast:** Includes morning refreshments and coffee.
- **Lunch:** Includes midday refreshments and coffee.
- **Dinner:** Includes evening refreshments and coffee.
- **Assessment:** Includes practical and theoretical assessments with feedback sessions.
- **Safety & Legal Briefing:** Includes mandatory safety and legal briefings.
- **Student Team Briefs:** Includes team briefs for specific tasks.
- **Assessment Preparation:** Includes preparation for upcoming assessments.
- **Free:** Includes free time and breaks.

**Version:** 20230516
What’s it like?
What’s it like?
Ultimate Truck Hacking Platform

Brings together useful hardware, tools, and operating systems for a complete, low-cost system for education, training, research and assessments for cybersecurity of heavy vehicles across multiple networking layers in the system.
Truck Networks

- Application Layer
- Network Layer
- Data Link / Transport Layer
- Physical Layer / CAN

- J1939 Message
- J1939 PDU
- CAN Frame

- ECU
- Gateway
Cybersecurity Threats in Commercial Vehicles

Truck Hacking: An Experimental Analysis of the SAE J1939 Standard
Authors: Yevgeniy Druzhinin, Dore Hase, Lee Miller, and Andrei Vorhersag (University of Michigan)

Abstract:
Consumer vehicles have been proven to be insecure; the addition of electronics to monitor and control vehicle functions have added complexity resulting in safety-critical vulnerabilities. Heavy commercial vehicles have also begun adding electronic control systems similar to consumer vehicles. We show how the openness of the SAE J1939 standard allows access to safety-critical attacks and that these attacks aren’t limited to one specific make, model, or industry.

We test our attacks on a 2006 Class 8 semi tractor and 2001 school bus. With these two vehicles, we demonstrate how simple it is to replicate the kinds of attacks used on consumer vehicles and that it is possible to use the same attack on other vehicles that use the SAE J1939 standard. We show safety-critical attacks that include the ability to accelerate a truck in motion, disable the driver’s ability to accelerate, and disable the vehicle’s engine brake. We conclude with a discussion for possibilities of additional attacks and potential remote attack vectors.

Open Access Media
USENIX is committed to Open Access to the research presented at our events. Papers and proceedings are freely available to everyone once the event begins. Any video, audio, and/or slides that are posted after the event are also free and open to everyone. Support USENIX and our commitment to Open Access.
Cybersecurity Threats in Commercial Vehicles

Security Shortcomings and Countermeasures for the SAE J1939 Commercial Vehicle Bus Protocol

Abstract:
The recent years, countless security concerns related to automotive systems were revealed either by academic research or real-life attacks. While current attention was largely focused on passenger cars, due to their ubiquity, the reported bus-related vulnerabilities are applicable to all industry sectors where the same bus technology is deployed, i.e., the CAN bus. The SAE J1939 specification extends and standardizes the use of CAN to commercial vehicles where security plays an even higher role. In contrast to empirical results that attest such vulnerabilities in commercial vehicles by practical experiments, here, we determine that existing shortcomings in the SAE J1939 specifications open road to several new attacks, e.g., Impersonation, denial of service (DoS), distributed DoS, etc. Taking the advantage of an industry-standard CANoe-based simulation, we demonstrate attacks with potential safety critical effects that are mounted while still conforming to the SAE J1939 standard specification. We discuss countermeasures and security enhancements by including message authentication mechanisms. Finally, we evaluate and discuss the impact of employing these mechanisms on the overall network communication.
Cybersecurity Threats in Commercial Vehicles

Exploiting Transport Protocol Vulnerabilities in SAE J1939 Networks

Rik Chatterjee, Subhojeet Mukherjee, Jeremy Daily (Colorado State University)

Modern vehicles are equipped with embedded computers that utilize standard protocols for internal communication. The SAE J1939 protocols running on top of the Controller Area Network (CAN) protocol is the primary choice of internal communication for embedded computers in medium and heavy-duty vehicles. This paper presents five different cases in which potential shortcomings of the SAE J1939 standards are exploited to launch attacks on in-vehicle computers that constitute SAE J1939 networks. In the first two of these scenarios, we validate the previously proposed attack hypothesis on more comprehensive testing setups. In the later three of these scenarios, we present newer attack vectors that can be executed on bench test setups and deployed SAE J1939 networks. For the purpose of demonstration, we use bench-level test systems with real electronic control units connected to a CAN bus. Additional testing was conducted on a 2014 Kenworth T20 Class 6 truck under both stationary and driving conditions. Test results show how protocol attacks can target specific ECUs. These attacks should be considered by engineers and programmers implementing the J1939 protocol stack in their communications subsystem.
Yocto Build

Hardware

Tools
What is Yocto?

➢ Definition: Yocto is a collaborative project that provides templates, tools, and methods to create custom Linux-based systems for embedded products.

➢ OpenEmbedded Build System: Utilizes the OpenEmbedded build framework, which comprises a collection of "recipes" detailing how software components are compiled and integrated.

➢ Layered Approach: Organizes functionalities in layers, allowing for better modularity and separation of concerns.

➢ Not Just Another Distribution: Yocto isn't a Linux distribution itself but a toolkit to develop distributions tailored for specific needs.

➢ Backed by the Linux Foundation: Ensuring consistent development and maintenance backed by industry professionals.
Why Choose Yocto?

➢ Customized Linux Systems: Yocto crafts a tailor-made OS, optimized for specific needs and hardware architectures.
➢ Layer Model: Yocto’s flexible layering system, allows developers to add or remove functionalities as needed.
➢ Rich Ecosystem: A vast network of available layers and configurations on the net accelerates development, especially with layers designed by board founders and manufacturers.
➢ Complete Granularity: Beyond just generating full system images, Yocto offers granularity in generating bootloader, kernel, filesystem, and toolchain, ensuring every component is precisely as intended.
➢ Learning Curve vs. Flexibility: While Yocto has a steeper learning curve, its power and flexibility are unparalleled. When a project requires in-depth customization, particularly for embedded systems aiming for efficiency, Yocto is often the optimal choice.
➢ Comparative Advantage: Compared to desktop distributions, Yocto is more tailored for specific product goals, especially when lightness, speed, and ultra-customization are priorities. While Buildroot offers simplicity, Yocto provides a broader spectrum of packages and capabilities.
Building with Yocto for UTHP

- Install Essential Packages: Tools and dependencies for Yocto.
- Clone Poky Repository: The heart of the Yocto Project.
- Navigate to Poky Directory: View and choose the relevant branch.
- Checkout 'kirkstone' Branch: The designated release for our project.
- Initialize Build Environment: Using oe-init-build-env within the 'poky' directory.
- Review & Modify 'local.conf': Adapt configurations to your project needs.
- Execute BitBake: Kickstarts the entire build process.
- Monitor Build Progress: Track the construction of your custom Linux distribution.
- Flash the image to the Beaglebone: Use ‘dd’ or and other tool.
Building with Yocto for UTHP

https://github.com/SystemsCyber/UTHP
The Yocto Built: Is It Complete?
Customizing the build: Recipes & Layers

➢ Recipes:
  ○ Individual instructions for building software components.
  ○ Specify source location, dependencies, and installation procedure.

➢ Layers:
  ○ Collections of related recipes and configurations.
  ○ Provide structure and modularity to the build process.

➢ Importance:
  ○ Easily add or remove software components.
  ○ Separate customizations or configurations.
  ○ Maintain third-party software, hardware adaptations, or proprietary software.

```
meta-<layer-name>
|-- recipes-<category>
    |-- <recipe-name>
    |    |-- <recipe-name>.bb
```
Customizing the build: Adding CAN

➢ Kernel Recipe Append Creation:
  ○ Create an append recipe specific for CAN: linux-yocto_5.15-can.bbappend.

➢ Specify Custom Configuration:
  ○ Add configurations in a separate can-support.cfg.
    ■ CONFIG_CAN=y
    ■ CONFIG_CAN_RAW=y
    ■ CONFIG_CAN_BCM=y

➢ Update Kernel Append Recipe for CAN:
  ○ Extend SRC_URI to include can-support.cfg.

➢ Integrate CAN-Utils:
  ○ Update image recipe: to include "can-utils"
Customizing the build: USB ETHERNET

➢ Kernel Recipe Append Creation:
  ○ Name: linux-yocto_5.15-usb.bbappend.
  ○ Purpose: Enable USB gadget features.
➢ Specify Custom Configuration:
  ○ Configuration Details:
    ■ CONFIG_USB_ETH=y
    ■ CONFIG_USB_G_NCM=m
    ■ CONFIG_USB_MASS_STORAGE=y
➢ Update Kernel Append Recipe:
  ○ Extend SRC_URI to include usb.cfg.
➢ Network Configuration:
  ○ Define static IP for USB Ethernet.
  ○ Update /etc/systemd/network/usb0.network on BeagleBone’s root filesystem.
➢ Host Configuration:
  ○ Manually set IP for the new network connection.
Customizing the build: Adding J1939

➢ Kernel Recipe Append Creation:
  ○ Create a J1939 specific append:
    ■ linux-yocto_5.15-j1939.bbappend.

➢ Specify Custom Configuration:
  ○ Define in j1939-support.cfg.
    ■ CONFIG_CAN_J1939=y

➢ Update Kernel Append Recipe for J1939:
  ○ Extend SRC_URI to include j1939-support.cfg.
Current Status

➢ USB Ethernet Integration:
  ○ Kernel Configuration: Successfully added and enabled.
  ○ Image Recipe: `usbinit` integrated.
  ○ Connectivity: Confirmed stable USB Ethernet connection via SSH.

➢ CAN Protocol Integration:
  ○ Kernel Configuration: CAN support enabled.
  ○ Image Recipe: `can-utils` integrated and functioning.
  ○ Communication: Successful test transmission and reception on CAN bus.

➢ J1939 Protocol Integration:
  ○ Kernel Configuration: J1939 support enabled.
  ○ Image Recipe: J1939 utilities and tools integrated.

```
ping 192.168.7.2
PING 192.168.7.2 (192.168.7.2) 56(84) bytes of data.
64 bytes from 192.168.7.2: icmp_seq=1 ttl=64 time=0.761 ms
64 bytes from 192.168.7.2: icmp_seq=2 ttl=64 time=0.354 ms
64 bytes from 192.168.7.2: icmp_seq=3 ttl=64 time=0.320 ms
64 bytes from 192.168.7.2: icmp_seq=4 ttl=64 time=0.313 ms
64 bytes from 192.168.7.2: icmp_seq=5 ttl=64 time=0.404 ms
64 bytes from 192.168.7.2: icmp_seq=6 ttl=64 time=0.361 ms
64 bytes from 192.168.7.2: icmp_seq=7 ttl=64 time=0.283 ms
64 bytes from 192.168.7.2: icmp_seq=8 ttl=64 time=0.356 ms
64 bytes from 192.168.7.2: icmp_seq=9 ttl=64 time=0.224 ms
```

```
--- 192.168.7.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 812ms
rtt min/avg/max/mdev = 0.224/0.370/0.761/0.140 ms
```

Please contact your system administrator.
Add correct host key in `/home/rik/.ssh/known_hosts` to get rid of this message.

```
remove with:
  `ssh-keygen -f "/home/rik/.ssh/known_hosts" -R "192.168.7.2"`
Host key for 192.168.7.2 has changed and you have requested strict checking.
Host key verification failed.
```

```
$ sudo rm -rf /home/rik/.ssh/known_hosts
$ ssh root@192.168.7.2
```

Warning: can't be established.
ED25519 key fingerprint is SHA256:AEsMnFnjye1lQnG6XKVHTCovp160xJ3hcCbbeh4E.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint]?)
```
Yocto Build  

Hardware  

Tools
Truck Networks
Hardware Overview

- CAN
- LIN
- PLC
- BeagleBone Black
- Real-time Clock
- Logic Analyzer
- Circuit Protections
- Connectors
- Safe Power-down
BeagleBone Black

➢ It's better than a Raspberry Pi for our application
➢ SBC for embedded systems and projects
➢ Specs: 1GHz ARM Cortex-A8, 512MB RAM, 4GB eMMC
  ○ Two CAN controllers inside
  ○ Two programmable real-time units (PRU)
  ○ Ample I/O (92 GPIO pins)
➢ Affordable, versatile, onboard storage, community support
➢ Used, verified, and tested in previous designs
UTHP PCB

Left: 3D view of the PCB layout showing component locations, beaglebone placement, and connector configurations.

Right: PCB layout of components.

NOTE: Designs shown are preliminary and are subject to change. The images seen are current as of October 19, 2023.
Controller Area Network (CAN)

- Requirement: J1939 supported
- The UTHP contains:
  - (4x) MCP2562FD CAN Transceivers
  - (2x) MCP2518FD CAN Controllers with external oscillator
  - (2x) 120 Ohm terminating resistor enable / disable switches
- Added 2 extra CAN interfaces to the BeagleBone
- Multiple connector interfaces
Left: MCP2518 CAN controller with a 40 MHz external oscillator, decoupling capacitors, and connections to the MCP2562FD CAN transceiver.

Bottom Left: MCP2562 transceiver circuit with decoupling capacitor.

Bottom Right: Single-pole single-throw switch allowing for enabling or disabling the 120 ohm terminating resistor.
Local Interconnect Network (LIN)

➢ Requirement: J1708/J1587 supported
➢ LIN implementation uses the MCP2003B chip
  ○ Stand-alone transceiver
  ○ Transient protection capacitors
Power Line Communication (PLC)

➢ 1st Option: Intellon SSC P485
  ○ 20-pin spread-spectrum carrier transceiver IC
  ○ Not in production, but able to purchase and documentation still available
  ○ Needs additional design work + hardware to implement
  ○ Cheapest option available
➢ Build breakout board to test
➢ PLC4TRUCKS resources
  ○ [https://github.com/TruckHacking/plc4trucksduck](https://github.com/TruckHacking/plc4trucksduck)
  ○ [https://nmfta.org/wp-content/media/2022/11/Power_Line_Truck_Hacking_2TOOLS4PLC4TRUCKS.pdf](https://nmfta.org/wp-content/media/2022/11/Power_Line_Truck_Hacking_2TOOLS4PLC4TRUCKS.pdf)

Above: Intellon SSC P485 typical application for PLC connection. Block diagram shows microcontroller, SSC P485 transceiver, filtering and signal conditioning, and coupler.

Right: Pinout of the 20-pin Intellon SSC P485.

Power Line Communication (PLC)

➢ 2nd Option: Semitech SM2400 PLC Module
  ○ Plug - N’ - Play : 4 pin connector
  ○ UART interface
  ○ Optional SPI / RS485 interfaces
  ○ Need to perform testing to compare against the Intellon SSC P485
  ○ Application notes, reference designs and software available as resources from the manufacturer
  ○ Not cost-friendly at ~$100 per unit

Power Line Communication (PLC)

➢ 3rd Option: Build our own “transceiver”
  ○ Design process and schematic shown below
    ■ Only able to perform amplitude shift keying (preamble), but not phase-shift keying
  ○ Using a filter: resonant frequency, Q-factor, and gain equations give component values

![Diagram of a signal processing circuit](https://www.electronics-tutorials.ws/filter/filter_7.html)

**Infinite Gain Multiple Feedback Active Filter**
- **UI7:**
  - **V+**
  - **V-**
- **SHDN:**
  - **R1:** 11.3k
  - **C1:** 180pF
  - **R2:** 1k
  - **C2:** 180pF

**Passive Low Pass Filter**
- **PLCIn:**
- **PLCFiltered:**
  - **R39:** 100k
  - **C34:** 0.1uF


- Maximum Gain, \( Av \) = \(- \frac{R_2}{2R_1} = -2Q^2\)
- Resonant Frequency, \( f_r \) = \( \frac{1}{2\pi \sqrt{R_1R_2C_1C_2}} \)
- Quality Factor, \( Q_{BP} \) = \( \frac{f_r}{BW_{(3dB)}} = \frac{1}{2\pi f_r} \sqrt{\frac{R_2}{R_1}} \)
J1708 / J1587

➢ Requirement: J1708 / J1587 supported
➢ Combination of inverters, resistors and transceiver capable of handling J1708 traffic
   ○ THVD1410DR transceiver has built-in ESD protection, low power consumption, bus failsafes and noise rejection
➢ Two J1708 / J1587 circuits
   ○ Seen below with decoupling capacitor
   ○ Connected to multiple output connectors

Primary J1708 Circuit
# Real-Time Clock + Security Chip

ATECC608A Security Module | MCP7940N Real Time Clock (RTC)
---|---
I2C communication | I2C communication
Secure boot support | Hours, Minutes, Seconds, Day of Week, Day, Month and Year
Hardware-based key storage | Battery-powered time keeping
Educational feature | Low-power
Future work of number generation | Timestamp when switching to battery

https://www.microchip.com/en-us/product/atecc608a  
https://www.microchip.com/en-us/product/mcp7940n
Safe Power-Down

➢ BeagleBone would often come unplugged from power / USB
➢ Led to issues with OS image - would become corrupted
  ○ Led to non-operational state for BeagleBone
  ○ Have to re-flash SD card and BeagleBone
  ○ If corruption happened enough times, BeagleBone doesn’t work anymore
➢ Need a way to maintain power for short period after power loss and shut down properly

https://learn.adafruit.com/li-ion-and-lipoly-batteries/voltages

Safe Power-Down

➢ Found BeagleBone “test points” for battery > connected large capacitor (5F) to store charge > developed script

Above: Circuit schematic containing capacitor for BeagleBone safe power-down.

Above: Supercapacitor circuit addition close-up view on TruckCape V4. The circuit contains a 5F capacitor, 10k resistor, and a small board used for connecting components.

Right: TruckCape V4 board design showing scale of supercapacitor addition to the device.
Safe Power-Down

- Created a service:
  - OS runs the service in the background
  - Polling register on the power-management chip onboard the BeagleBone
    - TPS65217C chip, status being read over I2C
    - Register 0x24
    - Status 0x80 shows power has switched to “battery” (supercapacitor)
- If 0x80 is seen, run the command “shutdown -h now”
  - This turns off the BeagleBone properly, without corrupting the image
- Tested in the lab and on our research truck
  - Success!
Connectors

➢ Banana jacks coming soon
  ○ Externally accessible
  ○ CAN, LIN, J1708
  ○ 3.3V, 5V, 12V, GND

➢ Deutsch 9
  ○ Custom cable connected to UTHP via Molex 10-position header

➢ PWM connector
  ○ Output signals with configurable duty cycles
  ○ Sensor simulation, driving external circuitry, etc…

Right: MikroeClick connector interface allowing for modular additions to the UTHP.

Above: DB9 cable for representation of the 9-pin UTHP connector

Above: Double-stacked DSUB-25 connector.
Circuit Protections

- Protect the BeagleBone during Boot
- LTV-247 Optoisolators for higher-voltage
- SN74LVCH16 transceivers / translators
  - Buffering BeagleBone GPIO pins
- Bi-directional ESD protection for logic inputs
- Numerous decoupling capacitors

Above: Optoisolator for higher-voltage inputs.

Above: Buffer used for BeagleBone GPIO pins.

Above: ESD protection IC’s for digital inputs.
Logic Analyzer

➢ 12-inputs connected through ESD protections > buffer > BeagleBone GPIO pins
➢ Utilizes SN74LVCH16 buffer
➢ 24 pin header interface:
  ○ 12 pins grounded
  ○ 4 high voltage pins
  ○ 8 pins for 0-5V logic

Left: Current logic analysis circuit implementation for the UTHP.

Right: Wurth 24-pin header for logic analysis.
Tools / Software (In the Works)

Recipes are being built for the following tools for the UTHP:

- Cmap
- Scapy
- Python-can-j1939
- TruckDevil
- CanCat
- Py-hv-networks
- Plc4trucksduck
- Pretty_j1939
- Pretty_j1587
- Sigrok
- Can2 Decoder
- Canmatrix
- Ipython3
- Tmux
- Jupyter-lab

Please suggest your favorite hacking tool so we can include it in the build.
Grateful Acknowledgement

Thank you to NMFTA for the support for this project.

Once completed, NMFTA members should have access to the UTHP.

Seeking beta testers to provide feedback.
Thank you