**User Manual**

**Central Engine Controller Setup:**

**1.Circuit Diagram:**

BeagleBone Black 3.3V power pin P9\_3 to TFT Vin, LED.

BeagleBone Black ground pin P8\_2 to TFT GND.

BeagleBone Black SCLK pin P9\_22 to TFT CLK.

BeagleBone Black MOSI pin P9\_18 to TFT MOSI.

BeagleBone Black CE0 pin P9\_17 to TFT CS.

BeagleBone Black pin P9\_15 to TFT D/C.

BeagleBone Black pin P9\_12 to TFT RST.

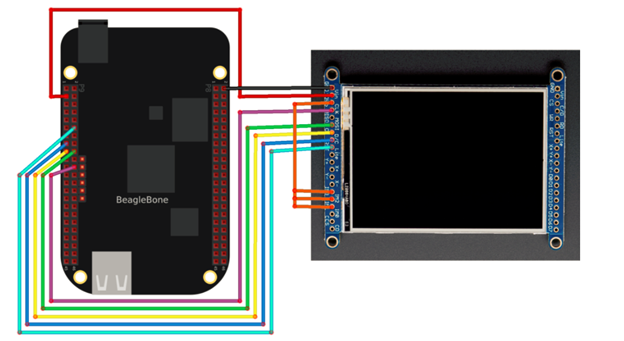


Figure 1 - Beaglebone Black To LCD connection

**2.Initialize LCD library:**

**Execute the following line in terminal:**

**sudo apt-get update**

**sudo apt-get install build-essential python-dev python-smbus python-pip python-imaging python-numpy git**

**sudo pip install Adafruit\_BBIO**

**cd ~**

**git clone https://github.com/adafruit/Adafruit\_Python\_ILI9341.git**

**cd Adafruit\_Python\_ILI9341**

**sudo python setup.py install**

**3. Initialize CAN BUS**

**Execute the following line in terminal:**

**Modprobe can**

**Modprobe can-dev**

**Modprobe can-raw**

**Config-pin -a P8\_13 gpio**

**Config-pin -a P8\_13 out**

**Config-pin -a P8\_13 low**

**Config-pin -a P9\_26 can**

**Config-pin -a P9\_24 can**

**Ip link set can1 up type can bitrate 250000 restart-ms 100**

**Ifconfig can1 up**

**4. Run the ECU script:**

**Download and Execute the following line in terminal:**

**Sudo python DisplaywithCan.py**

**The LCD and CAN BUS should running.**

**Reference:**

[**https://learn.adafruit.com/user-space-spi-tft-python-library-ili9341-2-8?view=all**](https://learn.adafruit.com/user-space-spi-tft-python-library-ili9341-2-8?view=all)

[**http://www.thomas-wedemeyer.de/beaglebone-canbus-python.html**](http://www.thomas-wedemeyer.de/beaglebone-canbus-python.html)

# Attack System

## Setup

* Plug the system into a 5v 1a power source
* Wait until the rgb led is green
* Connect the CANL and CANH wires to Pins 6 and 14 of an OBD-II port or the CANL and CANH breakout on the test systems.
* Press one of the 5 mode buttons to start an attack or diagnostic session.
* The session can be ended and brought back to the main menu by pressing the corresponding button again or power cycling the device.

## Modes

### Mode 1 - Data Logger

**Description**: Logs all CAN traffic to a text file.

**Use**: Log files can be played back over vcan to simulate a system and can also be graphed using the grapher.py script to find possible attack vectors.

**Physical**: Button 1, Dark Blue LED

### Mode 2 - DoS (Denial of Service)

**Description**: Sends packets of ID#000 with a data value of [0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00] with no time delay.

**Use**: Flooding the CAN bus with the highest priority and longest message in an attempt to stop network traffic. Some information does get through, but most is stopped by the attack.

**Physical**: Button 2, Red LED

### Mode 3 - ID Replication

**Description**: Takes ID, Data, and Sleep variables and will send packets with those constraints onto the network at the given sleep interval.

**Use**: Used to spoof nodes on the network and override their data. Examples would be showing incorrect information on clusters, triggering ABS, or steering a vehicle in park assist mode.

**Physical**: Button 3, Yellow LED

### Mode 4 - UDS (Unified Diagnostic Service)

**Description**: Sends a Diagnostic Session Request frame on a range of IDs to find ECUs that could be possible attack vectors in the system.

**Use**: A good starting point for complex systems like auto-driving or electric vehicles to narrow down IDs to check for.

**Physical**: Button 4, Light Blue LED

### Mode 5 - Corruption

**Description**: Sets an incorrect bitrate for the bus and forces a reinitialization every millisecond which causes the network to break.

**Use**: Can be used to test the operation of a vehicle if the CAN bus had been destroyed or disconnected.

**Physical**: Button 5, Magenta LED