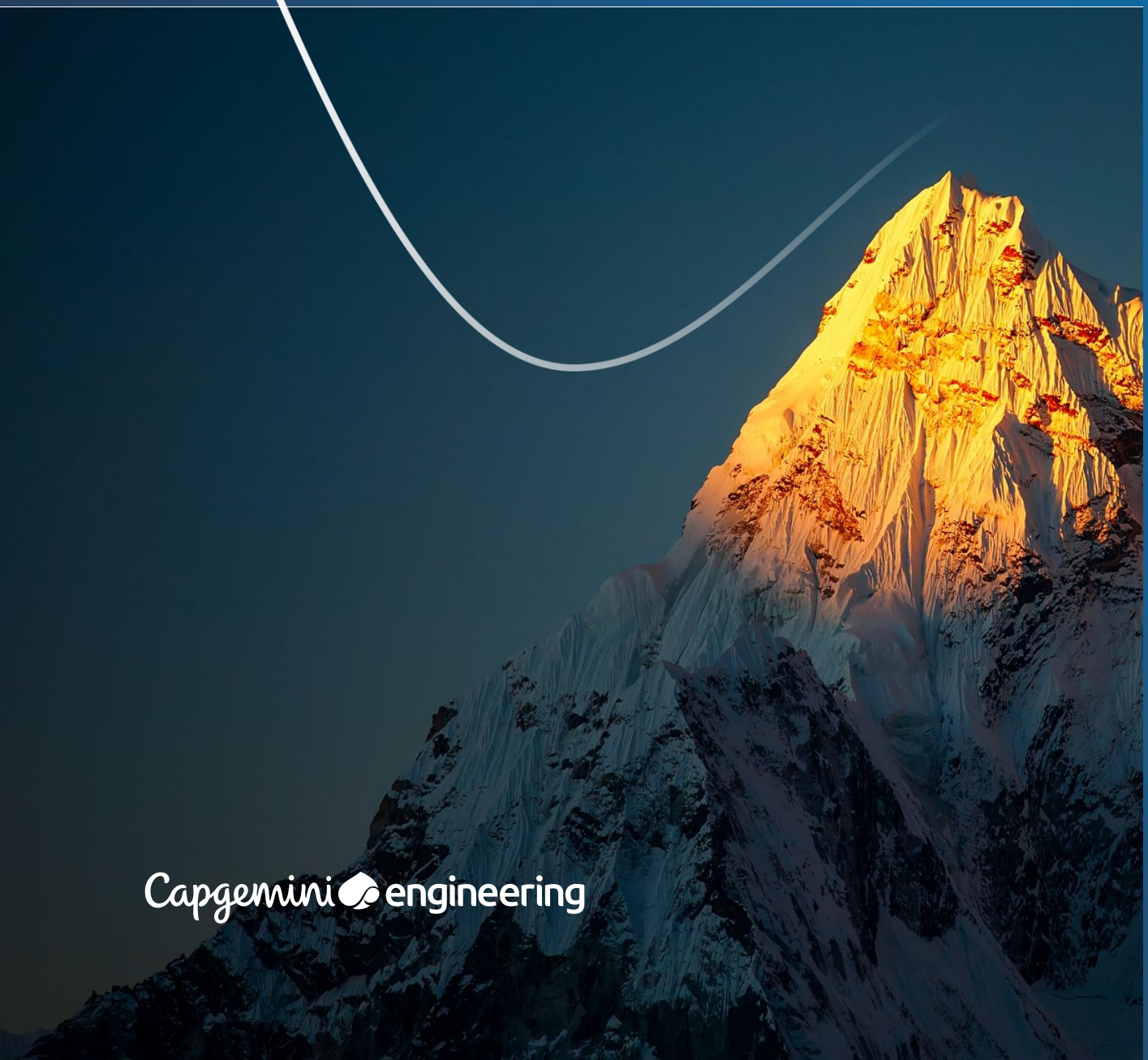
Chipset Activity – July 2025

Interfacing INA219 DC Current Sensor with NVIDIA JETSON NANO

Activity done by Sudharsan S



Capgemini engineering

Content

Introduction

This document provides a comprehensive technical overview of integrating the INA219 sensor with the NVIDIA Jetson Nano platform. The primary objective of this work is to enable real-time monitoring of voltage, current, and power at both the bootloader level and the OS level by leveraging U-Boot customization.

The activity covers four major tasks:

1. Building a Jetson Nano image using Jetson Disk Image Creator with NVIDIA's Linux for Tegra (L4T) BSP version 32.7.4.

2. Modifying U-Boot to display my name during the boot process.

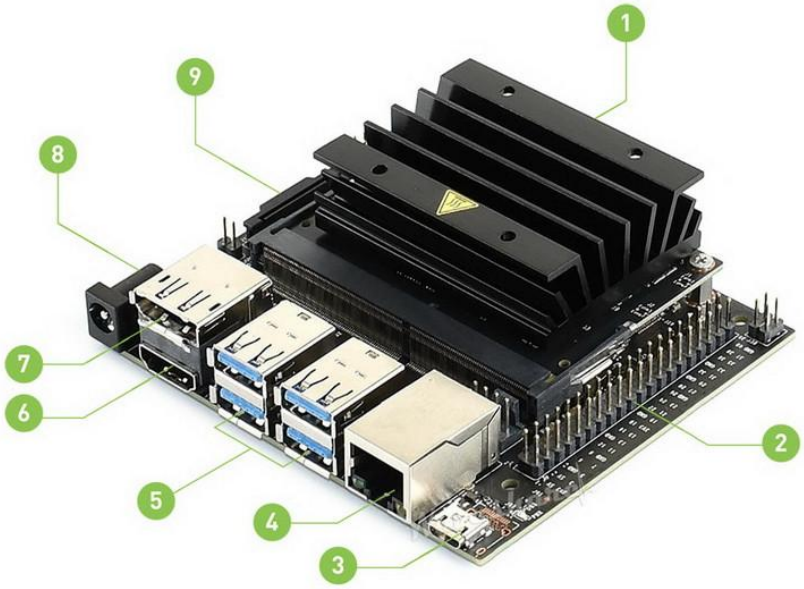
3. Developing and integrating a custom U-Boot command to read INA219 sensor vendor id and some registers.

4. Reading the sensor parameters from the Nvidia L4T Tegra custom-built Linux.

This implementation demonstrates embedded Linux development skills, including cross-compilation, bootloader customization, and I^2C peripheral interfacing.

NVIDIA Jetson Nano Overview

- **GPU:** NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
- **CPU:** Quad-core ARM Cortex-A57 MPCore processor
- **Memory:** 4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
- **Storage:** 16 GB eMMC 5.1
- **Operating System:** Linux for Tegra (L4T) based on Ubuntu
- **Connectivity:** Gigabit Ethernet, M.2 Key E
- **Display:** HDMI 2.0 and eDP 1.4
- **USB:** 4x USB 3.0, USB 2.0 Micro-B
- **Expansion:** 40-pin GPIO header (I²C, I²S, SPI, UART, GPIO), M.2 Key E slot
- **Power Options:** Micro-USB (5V 2A), DC Barrel Jack (5V 4A)



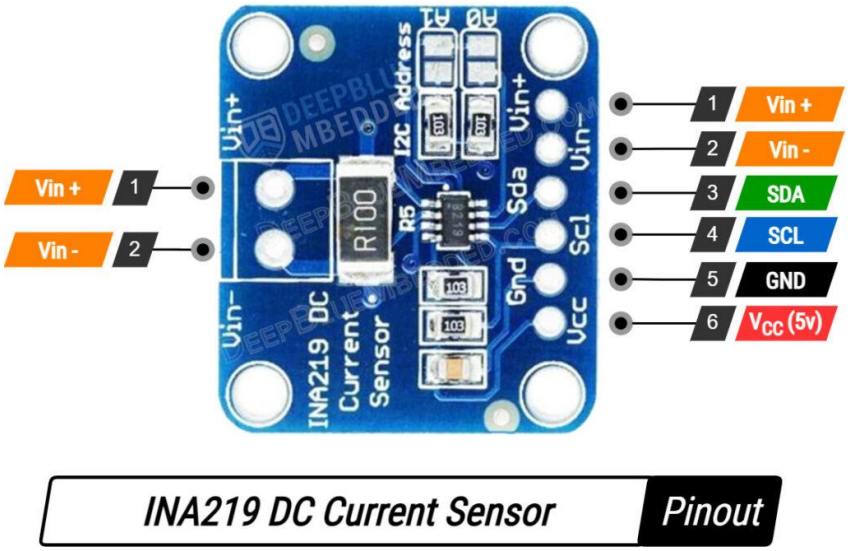
1. **Micro SD card slot:** insert a 16GB or larger TF card for main storage and writing system image
2. **40-pin expansion header**
3. **Micro USB port:** for 5V power input or for USB data transmission
4. **Gigabit Ethernet port:** 10/100/1000Base-T auto-negotiation
5. **4x USB 3.0 port**
6. **HDMI output port**
7. **DisplayPort connector**
8. **DC jack:** for 5V power input
9. **MIPI CSI camera connector:**

For more information kindly refer to the below document link

NV_Jetson_Nano_Developer_Kit_User_Guide.pdf

INA219 DC Current Sensor Specifications

- **Operational Voltage:** 3 V – 5.5 V
- **Operating Temperature:** –40 °C to +125 °C
- **Maximum Voltage:** 6 V
- **Bus Voltage Range:** 0 V – 26 V
- **Current Sensing Range:** ±3.2 A with ±0.8 mA resolution
- **Integrated Shunt Resistor:** 0.1 Ω, 1%, 2 W
- **Interface:** I²C or SMBus-compatible
- **Features:**
    - Zero-drift, bidirectional current/power monitor
    - Measures shunt voltage, bus voltage, current, and power
    - Programmable calibration for different shunt resistors



INA219 DC Current Sensor Pinout

Creating image for Jetson Nano

To build a full image for the NVIDIA Jetson Nano.

1. Click on the below link and download the L4T Driver Package (BSP) and the Sample Root Filesystem from the driver row of the Jetson nano

   https://developer.nvidia.com/embedded/linux-tegra-r3274

2. Set up the build environment

   ```
   $ sudo apt update
   $ sudo apt install -y build-essential bc git wget curl cpio python3 python3-pip libncurses5-dev libssl-dev flex bison libelf-dev
   ```

3. Open a terminal and create a new directory named Jetson. Copy the required files into this directory and extract them to prepare for further setup.

   ```
   $ mkdir ~/Jetson
   $ cd ~/Jetson
   $ tar xf Tegra_Linux_R32.7.4_aarch64.tbz2
   $ cd Linux_for_Tegra/rootfs
   $ sudo tar xf ../../Tegra_Linux_Sample-Root-Filesytem_R32.7.4_aarch64.tbz2
   $ cd ..
   $ sudo ./apply_binaries.sh
   ```

4. Use the Jetson Disk Image Creator tool, located in the `tools/` directory of the L4T package, to build a complete image for the NVIDIA Jetson Nano.

   ```
   $ sudo ./tools/jetson-disk-image-creator.sh -o Custom-Jetson.img -b jetson-nano -r 300
   ```

5. Format the SD card and use the following command to flash the image onto it. Replace x with the appropriate drive identifier. Use the lsblk command to determine the correct device name.

   ```
   $ sudo dd if=Custom-Jetson.img of=/dev/sdX bs=4M status=progress
   ```

6. After flashing, insert the SD card into the Jetson Nano. Connect an HDMI display, keyboard, and mouse to begin system configuration. Once the setup is complete, the OS will boot, and the system will be ready for use.

Customizing U-Boot

Customize the U-Boot to print my name during the booting process.

1. Clone the U-Boot repo.

```
$ git clone https://nv-tegra.nvidia.com/3rdparty/u-boot.git
```

2. Next, add name or any print statements within the main() or board_init () functions of one of the specified source files.

```
$ cd u-boot
$ nano common/board_f.c
$ nano common/board_r.c
$ nano common/main.c
$ nano board/nvidia/p3450-0000/p3450-0000.c
```

3. Download and install the cross-compiler if it is not already installed on the system.

```
$ sudo apt update
$ sudo apt install gcc-aarch64-linux-gnu
```

4. Compile and build the u-boot.

```
$ make distclean
$ export CROSS_COMPILE=aarch64-linux-gnu-
$ make p3450-0000_defconfig
$ make -j$(nproc)
```

5. Copy the generated binary files into the bootloader directory of the L4T (Linux for Tegra) package.

```
$ cp u-boot{,.bin,.dtb,-dtb.bin} ../bootloader/t210ref/p3450-0000/
$ cd ..
```

6. Before flashing U-Boot to the QSPI memory of the Jetson Nano, ensure the device is in recovery mode. To do this,
   - Short the 9th and 10th pin of the J50 header using jumper.
   - Connect a micro-USB to the jetson and to the host pc.
7. Verify if the jetson nano is in recovery mode by using the `lsusb` command.

```
$ lsusb | grep -I nvidia
```

8. Seeing a device entry similar to the below mentioned, successfully indicates that the Jetson Nano is in recovery mode.

```
Bus 001 Device 008: ID 0955:7f21 NVIDIA Corp. APX
```

9. Proceed to flash U-Boot into the QSPI memory of the Jetson Nano using the appropriate flashing script.

```
$ sudo ./flash.sh -x 0x21 jetson-nano-devkit mmcblk0p1
```

10. Optionally connect a USB-to-TTL converter to monitor the flashing process in detail via a serial console.

11. Once the boot process begins and reaches U-Boot, the print statement will be displayed, confirming that the custom modifications have been successfully applied.

```
U-Boot 2025.10-rc4-00029-gd33b21b7e261-dirty (Sep 24 2025 - 12:07:59 +0530)

SoC: tegra210
Model: NVIDIA Jetson Nano Developer Kit
Board: NVIDIA P3450-0000
DRAM:  4 GiB

My Custom u-boot
Core:  66 devices, 21 uclasses, devicetree: separate
MMC:   sdhci@700b0000: 1, sdhci@700b0600: 0
Loading Environment from SPIFlash... SF: Detected mx25u3235f with page size 256 Bytes, erase
size 4 KiB, total 4 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   No ethernet found.

CG:Sudharsan S
Hit any key to stop autoboot: 0
```

Creating command for INA219 sensor

To create a C program to read the vendor ID and registers of the INA219 sensor from the U-Boot.

1. Write a C program capable of reading the registers of the INA219 sensor. Ensure that all the required header files used in the program are available within the `u-boot/include/` directory to avoid compilation issues.

2. Copy the file to the `u-boot/common/` folder.

3. Modify the Make file and the Kconfig file in the `u-boot/common/` folder with the below mentioned modifications.

    - Update the Makefile in the `common/` directory.

      ```
      $ obj-y += ina219.o
      ```

    - Update the Kconfig file in the same directory.

      ```
      config CMD_INA219
          bool "Enable INA219 sensor readout"
          default y
      ```

    - Verify I2C is enabled in U-Boot.

      ```
      $ grep CONFIG_SYS_I2C=y configs/p3450-0000_defconfig
      ```

    - If not enabled add this line in the p3450-0000_defconfig file

      ```
      $ CONFIG_SYS_I2C=y
      ```

    - Update the p3450-0000_defconfig file from `configs/` directory by adding the below line.

      ```
      $ CONFIG_CMD_INA219=y
      ```

4. Compile U-Boot using the appropriate build commands to generate the updated bootloader with the integrated INA219 support.

    ```
    $ export CROSS_COMPILE=aarch64-linux-gnu-
    $ make distclean
    $ make p3450-0000_defconfig
    $ make -j$(nproc)
    ```

5. After compilation, copy the generated .dtb and binary files into the bootloader directory of the Jetson Nano to prepare for flashing.

```
$ cp u-boot{,.bin,.dtb,-dtb.bin} ../bootloader/t210ref/p3450-0000/
```

6. Before flashing U-Boot to the QSPI memory of the Jetson Nano, ensure the device is in recovery mode. To do this,
    - Short the 9th and 10th pin of the J50 header using jumper.
    - Connect a micro-USB to the jetson and to the host pc.
7. Verify if the jetson nano is in recovery mode by using the `lsusb` command.

    ```
    $ lsusb | grep -I nvidia
    ```

8. Seeing a device entry similar to the below mentioned, indicates that the Jetson Nano is successfully in recovery mode.

    ```
    Bus 001 Device 008: ID 0955:7f21 NVIDIA Corp. APX
    ```

9. Proceed to flash U-Boot into the QSPI memory of the Jetson Nano using the appropriate flashing script.

    ```
    $ sudo ./flash.sh -x 0x21 jetson-nano-devkit mmcblk0p1
    ```

10. Optionally, connect a USB-to-TTL converter to monitor the flashing process in detail via a serial console.
11. When U-Boot starts, interrupt the automatic boot process by pressing any key. This will open the U-Boot command prompt, where the presence of the custom sensor command can be verified by entering help and reviewing the list of available commands.
12. Enable the I2C interface and verify sensor detection.
    - List available I2C buses
      `$ i2c bus`
    - Select the appropriate I2C bus
      `$ i2c dev 2`
    - Probe the bus
      `$ i2c probe`
13. A reference image below illustrates the expected output during this process.

```
Tegra210 (P3450-0000) # i2c bus
Bus 2:  i2c@7000c400
Bus 3:  i2c@7000c500
Bus 4:  i2c@7000c700
Bus 0:  i2c@7000d000  (active 0)
   3c: generic_3c, offset len 1, flags 0
Tegra210 (P3450-0000) # i2c dev 2
Setting bus to 2
Tegra210 (P3450-0000) # i2c probe
Valid chip addresses: 40
```

14. After completing the setup, execute the custom command at the U-Boot prompt to read the INA219 sensor manufacturer ID and registers. This will confirm that the integration is successful. Below is an example of how the command is used in this setup.

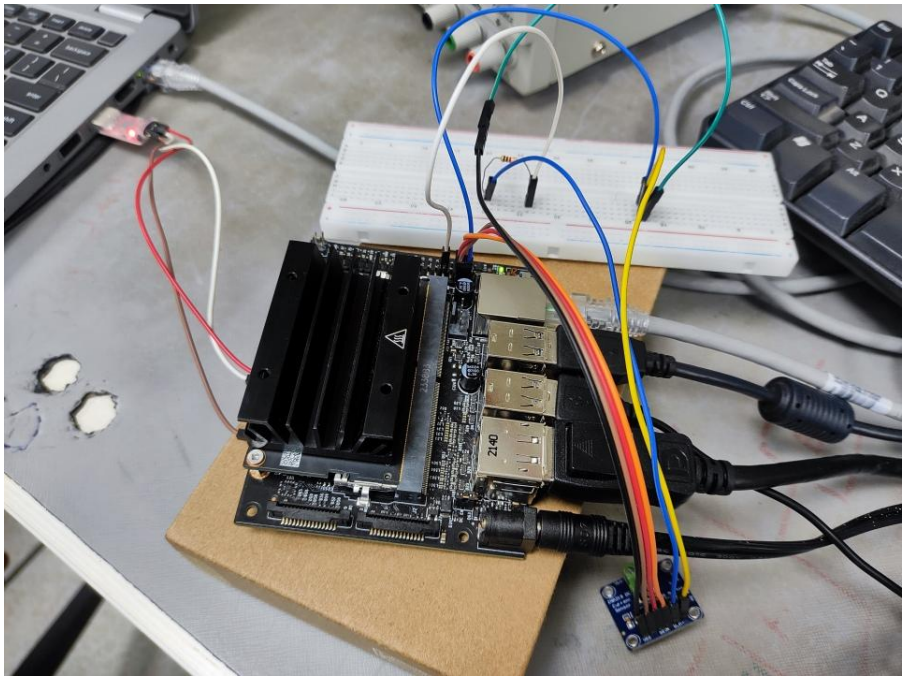
```
Tegra210 (P3450-0000) # ina219
Usage: ina219 <bus> <chip> <reg>

Tegra210 (P3450-0000) # ina219 2 0x40 0x00
INA219 [bus=2 chip=0x40] reg 0x00 = 0x399F
  RST=0, BRNG=1 (0=16V,1=32V), PG=3 (00=±40mV..11=±320mV)
  BADC=0x3, SADC=0x3, MODE=0x7 (7=cont shunt+bus)
Tegra210 (P3450-0000) # ina219 2 0x40 0x01
INA219 [bus=2 chip=0x40] reg 0x01 = 0xFFFD
Tegra210 (P3450-0000) # ina219 2 0x40 0x02
INA219 [bus=2 chip=0x40] reg 0x02 = 0x070A
  Bus voltage ≈ 900 mV
Tegra210 (P3450-0000) # ina219 2 0x40 0x03
INA219 [bus=2 chip=0x40] reg 0x03 = 0x0000
Tegra210 (P3450-0000) # ina219 2 0x40 0x04
INA219 [bus=2 chip=0x40] reg 0x04 = 0x0000
Tegra210 (P3450-0000) # ina219 2 0x40 0x05
INA219 [bus=2 chip=0x40] reg 0x05 = 0x0000
Tegra210 (P3450-0000) # ina219 2 0x40 0xFE
INA219 [bus=2 chip=0x40] reg 0xFE = 0x2000
Tegra210 (P3450-0000) # ina219 2 0x40 0xFF
INA219 [bus=2 chip=0x40] reg 0xFF = 0x3816
Tegra210 (P3450-0000) #
```

Reading INA219 Sensor parameters from custom OS

This section outlines the steps to read sensor data from the INA219 using the custom-built operating system.

1. Login to the custom OS running on the Jetson Nano.
2. Create a new C program that interfaces with the INA219 sensor to measure and display key parameters such as bus voltage, shunt voltage, current and power.
3. Connect the sensor to the Jetson Nano as shown below.



INA219	Jetson Nano
VCC	Pin 2 (5V)
GND	Pin 6 (GND)
SDA	Pin 3 (I2C SDA)
SCL	Pin 5 (I2C SCL)

- Vin + (INA219) → 5V Jetson Nano
- Vin – (INA219) → One terminal of resistor
- Another terminal of resistor → Jetson Nano GND

4. Verify I2C Bus and Device Detection.

```
$ i2cdetect -r -y 1
```

```
nvidia@nvidia:~$ i2cdetect -r -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

5. Compile the Program

```
$ gcc current.c -o current
```

6. Run the program.

```
$ ./current
```

7. The terminal will display real-time readings of bus voltage, shunt voltage, current, and power as measured by the INA219 sensor

```
Initializing INA219...
INA219 initialized successfully!

==============================
Bus Voltage   : 5.128 V
Shunt Voltage : 0.700 mV
Current       : 0.006 A
Power         : 0.033 W

==============================
Bus Voltage   : 5.128 V
Shunt Voltage : 0.700 mV
Current       : 0.006 A
Power         : 0.033 W

==============================
Bus Voltage   : 5.128 V
Shunt Voltage : 0.700 mV
Current       : 0.006 A
Power         : 0.033 W
```

8. Use a multimeter to cross-verify the sensor readings obtained from the program. This ensures the accuracy and reliability of the measurements reported by the INA219 sensor.

Conclusion

This document has demonstrated a complete workflow for integrating the INA219 current sensor with the NVIDIA Jetson Nano, from building a custom Linux image to customizing U-Boot and implementing sensor readouts at both bootloader and OS levels. Through this process, we explored key embedded development concepts including cross-compilation, I²C interfacing, bootloader modification, image building and sensor data acquisition. The successful implementation of a custom U-Boot command and real-time current monitoring from the operating system highlights the flexibility and power of the Jetson Nano platform for low-level hardware integration. This activity not only reinforces foundational embedded Linux skills but also serves as a practical reference for future sensor interfacing and system-level customization efforts.