User's Guide

AM62L-EVSE-DEV-EVM 用户指南

摘要

本文档涵盖了关键接口的硬件连接、软件配置和测试过程，包括电力线通信 (PLC)、CAN 总线和 UART (RS-485/RS-232) 通信。文中还介绍了如何配置和使用 EVerest 开源充电协议栈，该充电协议栈为 EV 充电基础设施提供了模块化平台。本指南提供了根据 IEC 61851-1 标准使用 MSPM0 微控制器（可处理关键模拟握手和安全功能）进行交流充电会话仿真的分步说明。本指南旨在帮助开发人员加快 EVSE 开发，并缩短产品上市时间，适应不同充电标准和地区规定。

内容

商标

1 简介

电动汽车供电设备 (EVSE) 系统变得越来越复杂，需要复杂的控制机制来支持多种充电标准和协议。为解决这一难题，AM62L-EVSE-DEV-EVM 参考设计提供了全面的前端控制器解决方案，可在电动汽车充电过程中用作中央通信模块 [1]。

此参考设计将 AM62L 处理器的处理能力与 MSPM0G3507 微控制器结合在一起，打造了一个多功能平台，支持各种全球标准的交流和直流充电，包括组合充电系统 (CCS)、Guobiao/Tuijian (GB/T) 和 Charge de Move (CHAdeMO) [1]。

该系统的核心是 AM62L 处理器，该处理器在 Linux 上运行 EVerest 开源软件充电栈，处理与电动汽车的数字通信。AM62L 支持用于后端通信的以太网和无线连接，以及用于人机界面 (HMI) 集成的显示功能 [1]。该处理器支持开发人员构建专注于用户体验和系统管理的复杂充电应用。

MSPM0G3507 微控制器与 AM62L 配合使用时，可用作前端控制器，管理电动汽车的关键模拟握手和安全功能。MSPM0 负责处理基本任务，例如控制引导信号、接近检测、监测充电连接器中的温度传感器 [1]。这种双处理器方法可确保上层通信协议和底层安全机制的可靠运行。

该设计包含多个通信接口（CAN、RS-485、RS-232 和以太网），可控制电源转换单元、外部计量器件和其他外设 [1]。借助这种全面的连接，系统可以与充电基础设施生态系统中的各种组件集成。

通过提供具有开源软件支持的完整参考平台，AM62L-EVSE-DEV-EVM 支持开发人员加快 EVSE 系统的开发，不仅缩短了产品上市时间，而且具有更高的灵活性，可适应不同的充电标准和地区规定。

请注意，AM62L-EVSE-DEV-EVM 在本文档的某些部分也称为 TIDA-010939。后者是 AM62L-EVSE-DEV-EVM 所采用的参考设计。

2 EVSE 开发平台的软件驱动程序概述

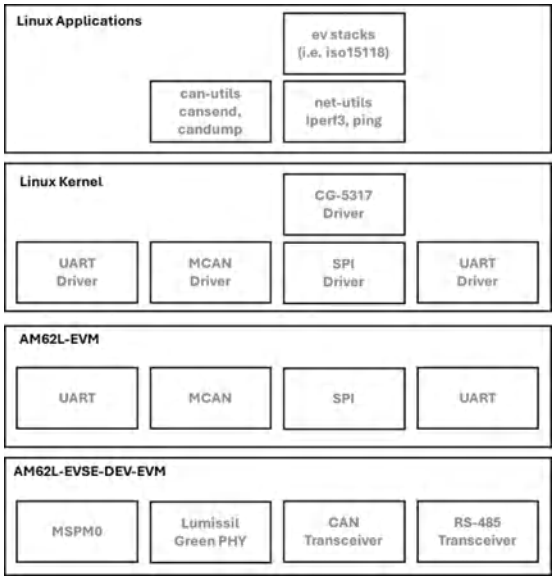


图 2-1. Linux 应用程序至硬件的流程

上图展示了一个分层架构，其中：

应用层：

- Linux 应用程序包含网络实用程序（即 net-utils，如 Iperf3、ping 等）
  - SD 卡和 TI Linux SDK 的映像中提供了这些实用程序。
- CAN 实用程序（即 can-utils，包括 cansend、candump）
  - SD 卡和 TI Linux SDK 的映像中提供了这些实用程序。
- EV 充电协议栈 (ISO15118)

驱动程序层：

- Linux 内核包含多个硬件驱动程序，用于此应用程序的驱动程序包括：
  - UART 驱动器
  - MCAN（控制器局域网）驱动程序
  - SPI 驱动器
  - CG-5317 驱动程序

硬件层：

- 该系统连接两个主要的硬件平台：
  - AM62L-EVM（带 UART、MCAN、UART 和 SPI 接口）
  - AM62L-EVSE-DEV-EVM（电动汽车供电设备开发板）

物理接口层：

- 硬件收发器和 PHY 组件（包括 RS-485 收发器、CAN 收发器、MSPM0、Lumissil 和 Green PHY）提供了到外部系统的物理接口 [1]

该架构演示了一个典型的 Linux 嵌入式系统。在该系统中，应用程序对内核进行系统调用，然后内核使用适当的设备驱动程序与底层硬件接口进行通信。这使得上层应用程序能够通过各种协议（CAN、UART、SPI）控制和通信，而无需直接管理硬件复杂性。

3 针对 AM62L-EVM 和 AM62L-EVSE-DEV-EVM 的测试设置建议

要测试 AM62L-EVSE-DEV-EVM 上的接口，建议将两个 AM62L EVM 连接到 AM62L-EVSE-DEV-EVM 电路板。这种设置将使用户能够进行测试，而无需昂贵的测试设备。例如，AM62L-EVSE-DEV-EVM 上 Lumissil Green PHY 设备的电力线通信 (PLC) 可以使用 IP 网络等以太网进行测试。CANBUS 可通过另一块 AM62L/AM62L-EVSE-DEV-EVM 开发板上集成的 canutils 应用程序，或通过外部 CANBUS 分析仪进行测试。最后，可在两块 AM62L/AM62L-EVSE-DEV-EVM 开发板之间使用 stty 命令对 RS-485 和 RS-232 接口进行测试。

建议的测试设置如下图所示。

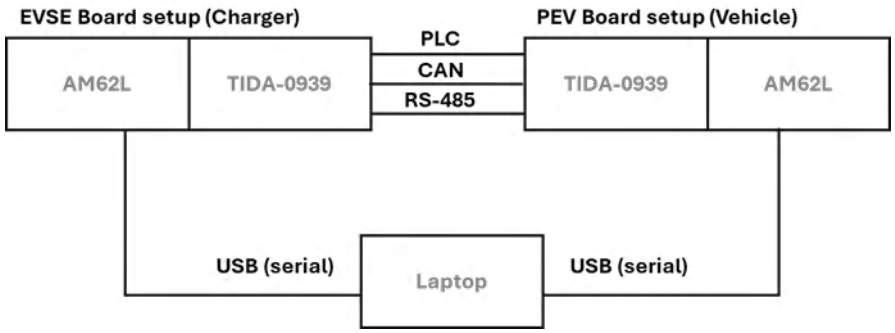


图 3-1. 建议的测试设置

可以使用电路板映像中提供的网络工具来测试 PLC 接口的基本运行情况。这些实用程序也是 TI Linux SDK 中的标准实用程序。在测试 PLC 接口时，需要将一个电路板指定为 EVSE，另一个指定为 PEV。作为指定过程中，PLC 端点有不同的配置，可使 SLAC 流程得以完成。本文档稍后将提供配置步骤以及两块电路板之间接口操作的分步指南。本文档后面还将提供 CAN 和 RS-485 / RS-232 接口的分步指南。

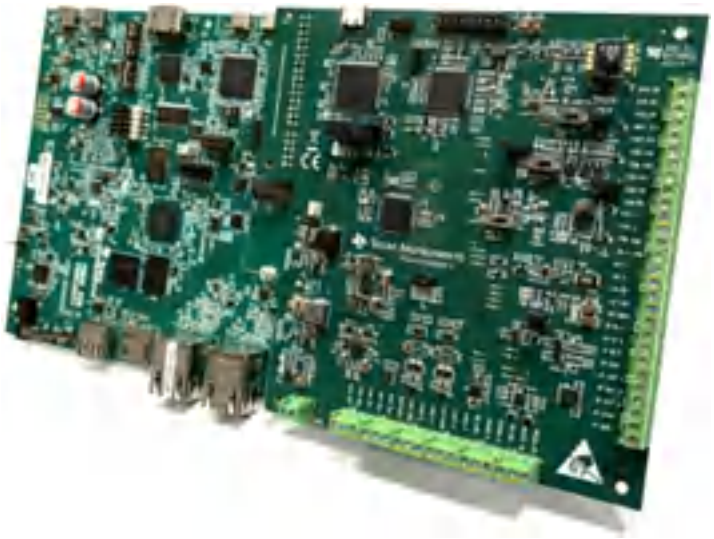
图中所示的笔记本电脑可任选操作系统（Linux、Windows 或 MAC）。要求连接到 AM62L EVM 上的 USB 串行连接。

借助此测试设置，您无需拥有或使用昂贵的 PLC、CAN 和 RS-485/RS-232 测试设备，即可熟悉 AM62L 与 AM62L-EVSE-DEV-EVM 开发板的组合使用。

4 AM62L-EVM 和 AM62L-EVSE-DEV-EVM 的开箱即用软件

本节将帮助您开始使用 AM62L 评估模块 (EVM)，使用 Linux SDK 和 AM62L-EVSE-DEV-EVM EV 充电参考设计来开发 EV 充电，也称为 EV 供电设备 (EVSE)。首先，请按照 AM62L EVM 的开箱即用或快速入门流程进行操作。完成相关步骤后，您将添加对 AM62L-EVSE-DEV-EVM 电路板的器件树覆盖支持。

假设 AM62L EVM 和 AM62L-EVSE-DEV-EVM 电路板已相互连接。



如本文档前面所述，为了获得最佳体验，建议进行双板设置。

请注意，阅读 AM62L-EVM 快速入门指南后，您需要下载映像。此链接对应的是要编程到 SD 卡的 wic 映像：

请注意，该映像中包含 EVerest 应用程序，默认为在启动时运行。该指南中的几个测试部分将提供有关如何停止 EVerest 应用程序的说明，以便执行接口测试演示。

请查看位于以下位置的 AM62L-EVM 快速入门指南：

https://dev.ti.com/tirex/local?id=TMDS62LEVM-QSG&packageId=PROCESSORS-DEVTOOLS

请注意，快速入门指南介绍了几个步骤，以下是唯一需要用到的步骤：

- 所需的电源、USB Type-C 和电流要求。
- 如何连接串行控制台和所需的电缆类型
- 引导模式选择（此应用将使用 SD 卡，并且是 AM62L EVM 的默认引导模式）
- 如何将 wic 映像编程到 SD 卡上
- 所需的串行控制台，以及如何在计算机上启用串行控制台
- 以太网（可选，此应用中不需要该接口）。
- 如何在引导后登录至 EVM。

以下步骤不会在该应用中使用：

- HDMI
- 鼠标
- 运行快速入门指南中提到的其他演示。

快速入门指南还提供了 EVM 无法启动时的一些基本调试步骤。

提供的 wic 映像将具有 AM62L-EVSE-DEV-EVM 电路板所需的驱动程序支持，因为该映像已经具有 AM62L-EVSE-DEV-EVM 子板所需的支持。本节介绍了在构建自己的发行版时，如何为 AM62L-EVSE-DEV-EVM 电路板添加必要的 DT 支持。

为了使 Linux 内核能够访问 AM62L-EVSE-DEV-EVM 上的接口，将关于 AM62L-EVSE-DEV-EVM 的支持添加到位于 SD 卡引导分区中的名为 uEnv.txt 的文件中。使用 USB SD 读卡器，将读卡器插入 Linux 计算机或 Windows 计算机。您将编辑位于引导分区中的 uEnv.txt 文件。

需要将此文件 k3-am62l3-evm-tida-010939.dtbo 添加到文件中的 name_overlays 行。

SDK 用户指南中的这一部分提供了有关向 uEnv.txt 文件添加叠加层的更多背景信息。

https://software-dl.ti.com/processor-sdk-linux/esd/AM62LX/11_01_16_13/exports/docs/linux/How_to_Guides/Target/How_to_enable_DT_overlays_in_linux.html

其他资源：

- TIDA-010939 设计指南
- 链接到 TIDA-010939 HW 用户指南
- AM62L 技术参考手册
- Linux SDK 用户指南
- MSPM0-SDK
- TI E2E 支持论坛

5 EVerest：开源电动汽车充电基础设施

概述

EVerest 是由 Pionix GmbH 开发的适用于 EV（电动汽车）充电基础设施的开源软件栈。它代表了一种灵活的模块化平台，旨在满足电动交通行业不断增长的需求。

主要特性

- 模块化架构：采用基于组件的设计构建，允许开发人员轻松自定义和扩展功能
- 开源：在 Apache 2.0 许可证下发布，促进协作和创新
- 不受硬件限制：跨不同的充电硬件平台工作
- 可互操作：支持 OCPP（开放充电协议）等行业标准
- 可扩展：适用于简单的住宅充电器和复杂的商业充电桩

应用

EVerest 可作为各种充电解决方案的基础：

- 家庭充电桩
- 公共充电基础设施
- 车队充电管理
- 通过可再生能源集成，实现智能充电
- 车辆到电网 (V2G) 应用

优势

- 缩短开发时间：预先构建的模块，可加快产品上市
- 面向未来：模块化设计允许随着技术的发展轻松更新
- 供应商独立性：避免专有锁定
- 成本效益：利用开放源代码降低开发成本
- 社区支持：以不断增长的开发人员和行业合作伙伴生态系统为后盾

Pionix 简介

Pionix GmbH 是 EVerest 平台背后的公司，专注于通过开源解决方案加快向可持续出行方案的过渡。该公司提供与 EVerest 平台相关的商业支持、咨询和定制开发服务。

6 测试 PLC 通信

测试 AM62L 和 AM62L-EVSE-DEV-EVM 电路板组合的电力线通信 (PLC) 的最佳测试方法，是将两个 AM62L EVM 与相应的 AM62L-EVSE-DEV-EVM 板连接在一起。这种测试设置方法是是演示与 PLC 链路对端通信的最简便方式。本节是为了帮助理解此类设置。这是推荐设置部分提供的示意图，作为快速参考，展示了双板配置。

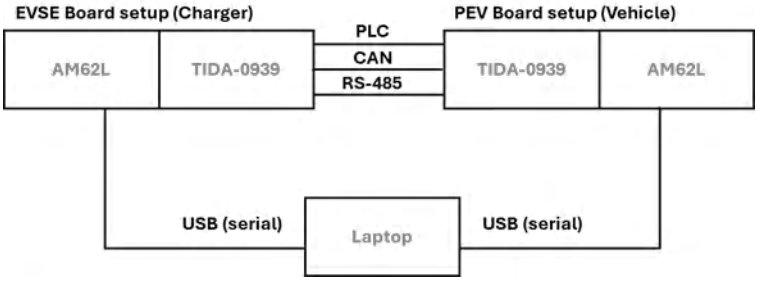


图 6-1. 双板设置

左侧的电路板设置视为充电器，右侧的电路板设置视为 EV。当您阅读本节时，将为您提供在 EVSE 或 PEV 上执行步骤的上下文。

以下说明需要用到本文档的“开箱即用”部分中定义的 SD 卡映像。如前文所述，有一个器件树源叠加层 (Dtso) 文件描述了 AM62L-EVSE-DEV-EVM 电路板上的接口，该文件以器件树二进制文件 (dtbo) 的形式提供。AM62L-EVSE-DEV-EVM 电路板的 dtbo 具有必要的 PLC 接口配置。此接口将位于 spi0 接口上，该接口将成为用于通信的 seth0 接口。该代码假定 dtbo 已添加到 SD 卡引导分区中的 uEnv.txt 文件中。U-Boot 会在系统引导序列期间读取此文件。

要启用 Lumissil PLC，必须先为 Lumissil PLC 通电，然后再重置。AM62L 软件配置为在启用 SPI 通信之前复位 PLC。因此，在启动 AM62L-EVM 之前，务必为 AM62L-EVSE-DEV-EVM（承载 PLC）上电。

验证 CG-5317 功能的第一步是验证驱动程序是否已初始化。使用这些命令来验证 CG-5317 是否已正确配置。该命令将检查 Lumissil 驱动程序的 systemd 服务的启动状态。屏幕截图显示接口已成功启用。应在 EVSE 和 PEV 电路板设置上执行此步骤。

root@am62lxx-evm:~# systemctl status cg5317-bringup

```
root@am62lxx-evm:~#
root@am62lxx-evm:~# systemctl status cg5317-bringup
* cg5317-bringup.service - cg5317 Interface Bringup Service
     Loaded: loaded (/etc/systemd/system/cg5317-bringup.service; enabled; preset: disabled)
     Active: inactive (dead) since Fri 2025-10-24 06:16:43 UTC; 2h 22min ago
    Process: 610 ExecStart=/usr/sbin/ip link set seth0 up (code=exited, status=0/SUCCESS)
   Main PID: 610 (code=exited, status=0/SUCCESS)
        CPU: 43ms

Oct 24 06:16:43 am62lxx-evm systemd[1]: Starting cg5317 Interface Bringup Service...
Oct 24 06:16:43 am62lxx-evm systemd[1]: cg5317-bringup.service: Deactivated successfully.
Oct 24 06:16:43 am62lxx-evm systemd[1]: Finished cg5317 Interface Bringup Service.
root@am62lxx-evm:~#
```

启动检查后，下一步是验证接口是否已初始化。

root@am62lxx-evm:~# systemctl status cg5317-host

```
root@am62lxx-evm:~# systemctl status cg5317-host
● cg5317-host.service - cg5317 Host Loading Service
     Loaded: loaded (/etc/systemd/system/cg5317-host.service; enabled; preset: disabled)
     Active: active (running) since Fri 2025-10-24 06:16:43 UTC; 2h 22min ago
    Process: 632 ExecStart=/root/examples/host_loading_service/host_loading_service -g gpiochip0 -o 39 (code=exited, status=0/SUCCESS)
   Main PID: 636 (host_loading_se)
      CPU: 330ms
     CGroup: /system.slice/cg5317-host.service
             └─636 /root/examples/host_loading_service/host_loading_service -g gpiochip0 -o 39

Oct 24 06:16:43 am62lxx-evm host_loading_service[636]: modem status file = /sys/devices/lms_eth2spi/modem_status
Oct 24 06:16:43 am62lxx-evm host_loading_service[636]: got modem_notification
Oct 24 06:16:43 am62lxx-evm host_loading_service[636]: [350012250] FW LOAD API: Entry
Oct 24 06:16:43 am62lxx-evm systemd[1]: Started cg5317 Host Loading Service.
Oct 24 06:16:43 am62lxx-evm host_loading_service[636]: [350012261] FW LOAD API: Com channel opened
Oct 24 06:16:43 am62lxx-evm host_loading_service[636]: [350012269] FW LOAD API: Device queried
Oct 24 06:16:43 am62lxx-evm host_loading_service[636]: [350012283] FW LOAD API: Com channel re-opened, starting fw loading
Oct 24 06:16:50 am62lxx-evm host_loading_service[636]: [350019232] FW LOAD API: Finished sending fw loading commands -> waiting for FW up
Oct 24 06:16:53 am62lxx-evm host_loading_service[636]: [350021857] FW LOAD API: Query reply received and FW is running
Oct 24 06:16:53 am62lxx-evm host_loading_service[636]: [350021862] FW LOAD API: Com channel closed
root@am62lxx-evm:~#
```

上图显示 cg5317 接口已成功初始化。

确认 cg5317 初始化成功的最后一个步骤是使用此命令验证接口是否已启动并位于网络接口列表中。

root@am62lxx-evm:~# ifconfig seth0

```
root@am62lxx-evm:~# ifconfig seth0
seth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::216:e8ff:fe99:de9b  prefixlen 64  scopeid 0x20<link>
        ether 00:16:e8:99:de:9b  txqueuelen 100  (Ethernet)
        RX packets 362  bytes 10456 (10.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 454  bytes 477413 (466.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@am62lxx-evm:~#
```

如果这些步骤成功，则可以继续进行配置以测试通信情况。

为了在 EVSE 和 EV 之间设置链路，必须运行脚本，将一个电路板配置为 EVSE，并将电路板组合配置为 EV。如上面的方框图所示，选择一个电路板组合作为 EVSE，另一个电路板作为 EV。从现在开始，我们将称之为 PEV，以匹配文件系统中的二进制文件。

要配置 PEV 板，请执行以下命令：

root@am62lxx-evm:~# cp /lib/firmware/spi_sta_config.bin /lib/firmware/config.bin

```
root@am62lxx-evm:~#  cp /lib/firmware/spi_sta_config.bin /lib/firmware/config.bin
root@am62lxx-evm:~#
```

root@am62lxx-evm:~# reboot

要配置 EVSE 板，请执行以下命令：

root@am62lxx-evm:~# cp /lib/firmware/spi_cco_config.bin /lib/firmware/config.bin

```
root@am62lxx-evm:~# cp /lib/firmware/spi_cco_config.bin /lib/firmware/config.bin
root@am62lxx-evm:~#
```

root@am62lxx-evm:~# reboot

重新引导完成后

在 PEV 板上运行：

root@am62lxx-evm:~# pev -p /etc/pev.ini -i seth0 &

在 EVSE 板上运行：

root@am62lxx-evm:~# evse -p /etc/evse.ini -i seth0 &

这些命令将启动 PEV 和 EVSE 板之间的 SLAC 序列。SLAC (信号电平衰减表征) 是一种用于电力线通信 (PLC) 的协议，尤其是在 HomePlug Green PHY 等标准中。在 EVSE 和 PEV 器件之间建立连接需要此初始化过程，通过连接两个系统的线缆进行通信。

此过程完成后，接口将为 IP 网络做好准备。SLAC 过程完成后，您应该会看到 EVSE 和 PEV 平台上都在充电的消息。下图展示了该情况。

```
evse -p /etc/evse.ini -i seth0 &
[3] 4784
root@am62lxx-evm:~# evse: evse_cm_set_key: --> CM_SET_KEY.REQ
evse: evse_cm_set_key: <-- CM_SET_KEY.CNF
evse: UnoccupiedState: Listening ...
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ
evse: evse_cm_slac_param: --> CM_SLAC_PARAM.CNF
evse: UnmatchedState: Sounding ...
evse: evse_cm_start_atten_char: <-- CM_START_ATTEN_CHAR.IND
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (0)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (0)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (1)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (1)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (2)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (2)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (3)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (3)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (4)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (4)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (5)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (5)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (6)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (6)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (7)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (7)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (8)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (8)
evse: evse_cm_mnbc_sound: <-- CM_MNBC_SOUND.IND (9)
evse: evse_cm_mnbc_sound: <-- CM_ATTEN_PROFILE.IND (9)
evse: evse_cm_atten_char: --> CM_ATTEN_CHAR.IND
evse: evse_cm_atten_char: <-- CM_ATTEN_CHAR.RSP
evse: UnmatchedState: Matching ...
evse: evse_cm_slac_match: <-- CM_SLAC_MATCH.REQ
evse: evse_cm_slac_match: --> CM_SLAC_MATCH.CNF
evse: MatchedState: Connecting ...
evse: MatchedState: waiting for pev to settle ...
evse: MatchedState: Charging (0) ...
```

现在，需要在 EVSE 和 PE 板上分配 IP 地址。对于 PEV 板，请运行以下命令：

ifconfig seth0 192.168.1.2/24 up

对于 EVSE 板，请运行以下命令：

ifconfig seth0 192.168.1.1/24 up

现在，电路板已准备好进行 ip 网络连接：

在 PEV 电路板上运行：

ping 192.168.1.1 (几次成功 ping 后，按 ctrl-c 退出)

在 EVSE 电路板上运行：

ping 192.168.1.2 (几次成功 ping 后，按 ctrl-c 退出)

现在，运行 iperf 命令来测试跨链路的移动数据

在 EVSE 上运行 iperf3 服务器命令：

iperf3 -s -i 2

在 PEV 板上运行 iperf3 客户端命令：

iperf3 -c 192.168.1.1 -t 20

20 秒后，PEV 上的 iperf3 客户端命令将停止。您应该会看到一个平均低于 1Mbps 的网络吞吐量。

测试到此结束。

7 测试 CAN 通信

建议将两个 AM62L 与配套的 AM62L-EVSE-DEV-EVM 电路板结合使用，并将 main_mcan0 接口连接在一起。与本文档中的其他部分一样，测试将按照此图中所示从 EVSE 和 PEV 的角度执行。

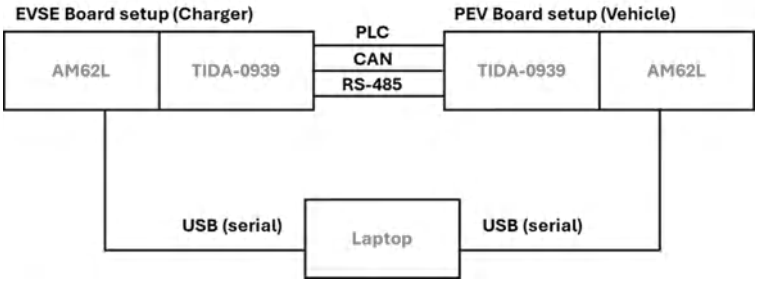


图 7-1. EVSE 和 PEV 角度

首先需要注意的是，默认 SDK 附带了 MCAN 支持，配置到 Linux 内核中。接下来需要指出的是，需要在器件树源文件中启用 MCAN 接口。如本文档前面所述，有一个器件树源叠加层 (dtso) 文件，它描述了 AM62L-EVSE-DEV-EVM 电路板上的接口，该文件以器件树二进制文件 (dtbo) 的形式提供。AM62L-EVSE-DEV-EVM 电路板的 dtbo 具有必要的 MCAN 接口配置。该代码假定 dtbo 已添加到 SD 卡引导分区中的 uEnv.txt 文件中。U-Boot 会在系统引导序列期间读取此文件。

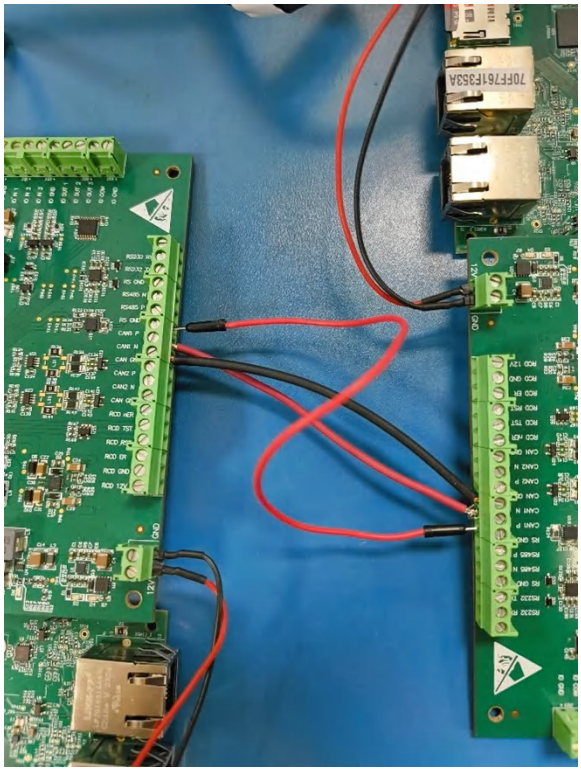


图 7-2. 连接两个板 CAN 接口

请注意用于连接两个电路板的导线。请注意，连接 CAN 接口时需要使用端接电阻器，而连接器和电路板上 CAN 收发器之间的线路上已经存在这些端接电阻器。

AM62L-EVSE-DEV-EVM 上的 CAN1 收发器直接连接到 AM62L 的 main_mcan1 接口。要在 AM62L 的第二个 CAN 接口与 AM62L-EVSE-DEV-EVM 上的第二个收发器之间建立连接，必须在两块电路板之间连接一根导线。

AM62L MCAN0 接口走线至位于 EVM 上的 J16 接头。要将 MCAN0 接口连接到收发器，应连接以下引脚：

表 7-1. J16 引脚

TIDA-010939 - J29		AM62L-EVM - J16	功能
引脚 1	更改为	引脚 2	TX
引脚 2	更改为	引脚 3	RX
引脚 3	更改为	引脚 4	GND

如果应使用 AM62L 的 MCAN2 接口，请将 EVM 的 J18 与 AM62L-EVSE-DEV-EVM 上的 CAN2 接头连接：

表 7-2. J18 引脚

TIDA-010939 - J29		AM62L-EVM - J18	功能
引脚 1	更改为	引脚 2	TX
引脚 2	更改为	引脚 3	RX
引脚 3	更改为	引脚 4	GND

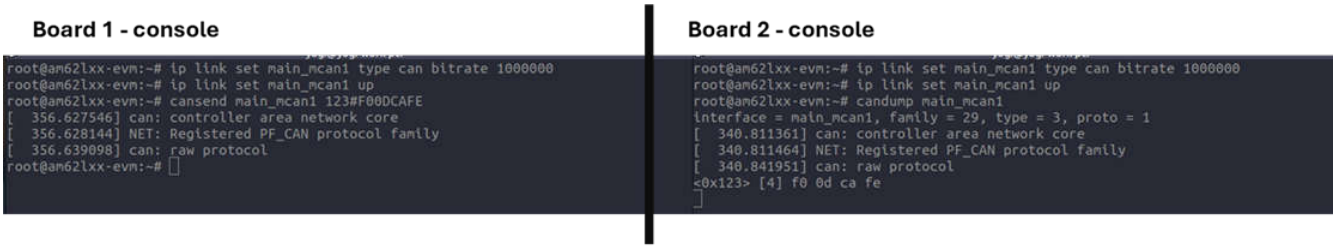
有关 TI Linux SDK 中 MCAN 驱动程序的更多信息，请参阅此链接。https://software-dl.ti.com/processor-sdk-linux/esd/AM62X/11_01_05_03/exports/docs/linux/Foundational_Components/Kernel/Kernel_Drivers/MCAN.html

MCAN 接口配置使用称为 ip 命令的网络实用程序进行，用于管理系统网络配置的各个方面。请注意，MCAN 接口是系统网络栈的一部分。这些命令位于 TI Linux SDK 的默认文件系统中。

MCAN 接口可配置为 CAN 和 CAN-FD 两种不同模式之一。对于 CAN 模式，请执行以下步骤：

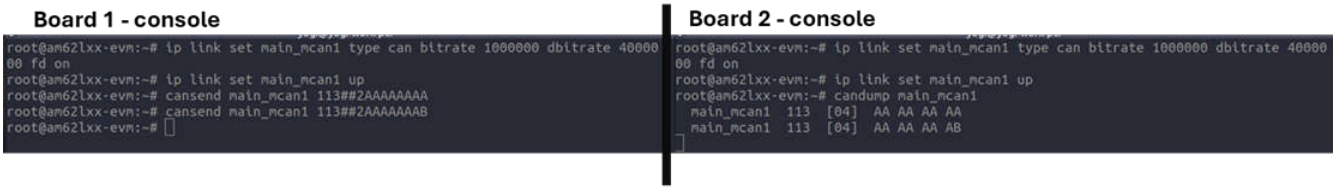
ip link set main_mcan0 type can bitrate 1000000

下面的截屏显示了先对 CAN 进行初始化，再检查接口列表以确认 can2 接口及其状态的操作顺序。这里显示了两种方法：基于 ip-route2 的方法和基于 ifconfig 与 net-utils 的方法。这两种方法都可以用于检查接口状态。



对于 CAN-FD，请执行以下步骤：

ip link set main_mcan0 type can bitrate 1000000 dbitrate 4000000 fd on



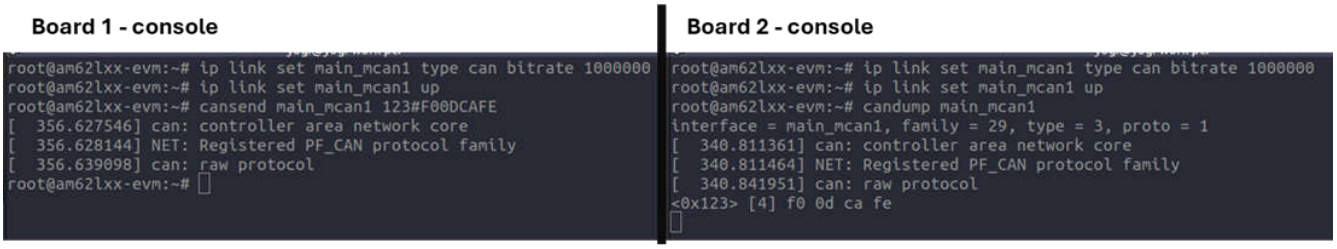
初始化接口后，使用 ip 命令来启动接口。

ip link set main_mcan0 up

若要发送帧，应使用以下命令。对于在 CAN 模式下发送帧：

cansend main_mcan0 123#F00DCAFE

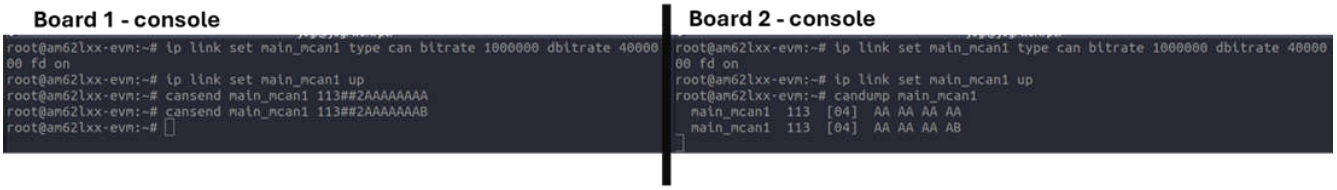
请注意，电路板 2 控制台转储输出与使用 cansend 从电路板 1 发送的输出匹配。



对于在 CAN-FD 模式下发送帧

cansend main_mcan0 113##2AAAAAAAA

candump main_mcan0



请注意，电路板 2 控制台转储输出与使用 cansend 从电路板 1 发送的输出匹配。

有关更多信息，请访问上面链接中指向 MCAN Linux 内核用户指南的链接。

8 测试 UART 通信 AM62L-EVM 和 AM62L-EVSE-DEV-EVM

测试 AM62L 和 AM62L-EVSE-DEV-EVM 电路板组合的 UART 通信的最佳测试方法，是将两个 AM62L EVM 与相应的 AM62L-EVSE-DEV-EVM 电路板连接在一起。这种测试设置方法是是演示与 UART 链路对端通信的最简便方式。本节是为了帮助理解此类设置。这是推荐设置部分提供的示意图，作为快速参考，展示了双板配置。

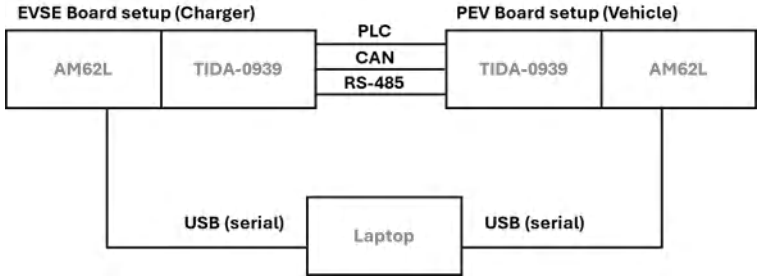


图 8-1. 双板设置

对于 RS485 测试：

使用 2 块 AM62L + AM62L-EVSE-DEV-EVM 电路板（我们将称之为电路板 A 和电路板 B）。按照下表连接 2 个电路板。

表 8-1. RS485 测试

电路板 A		电路板 B
RS485 N	ßà	RS485 N
RS485 P	ßà	RS485 P
RS 接地	ßà	RS 接地

请参阅下表了解更多详情。

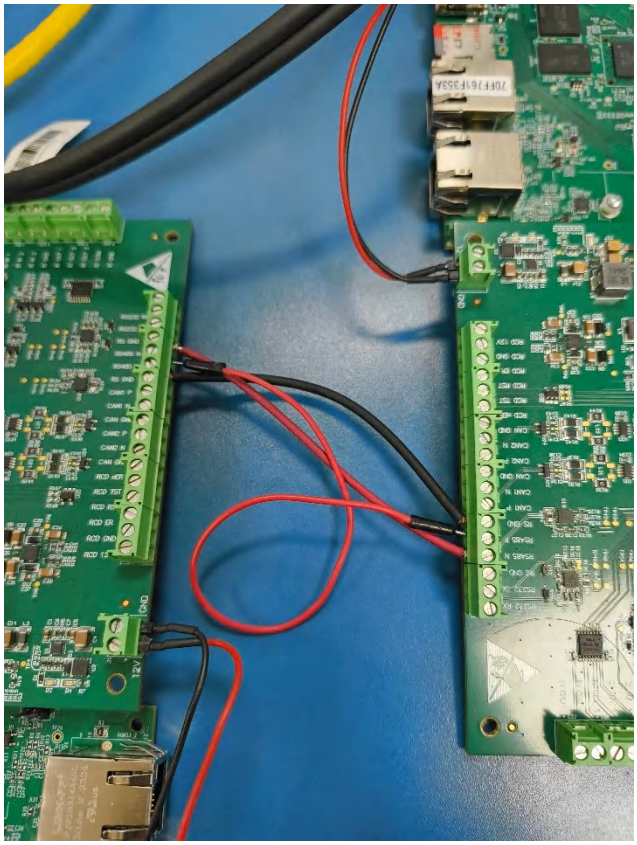


图 8-2. RS485 测试

对于 RS232 测试：

使用 2 块 AM62L + AM62L-EVSE-DEV-EVM 电路板（我们将称之为电路板 A 和电路板 B）。按照下表连接 2 个电路板。

表 8-2. RS232 测试

电路板 A		电路板 B
RS232 TX	ßà	RS232 RX
RS232 RX	ßà	RS232 Tx
RS 接地	ßà	RS 接地

请参阅下表了解更多详情。

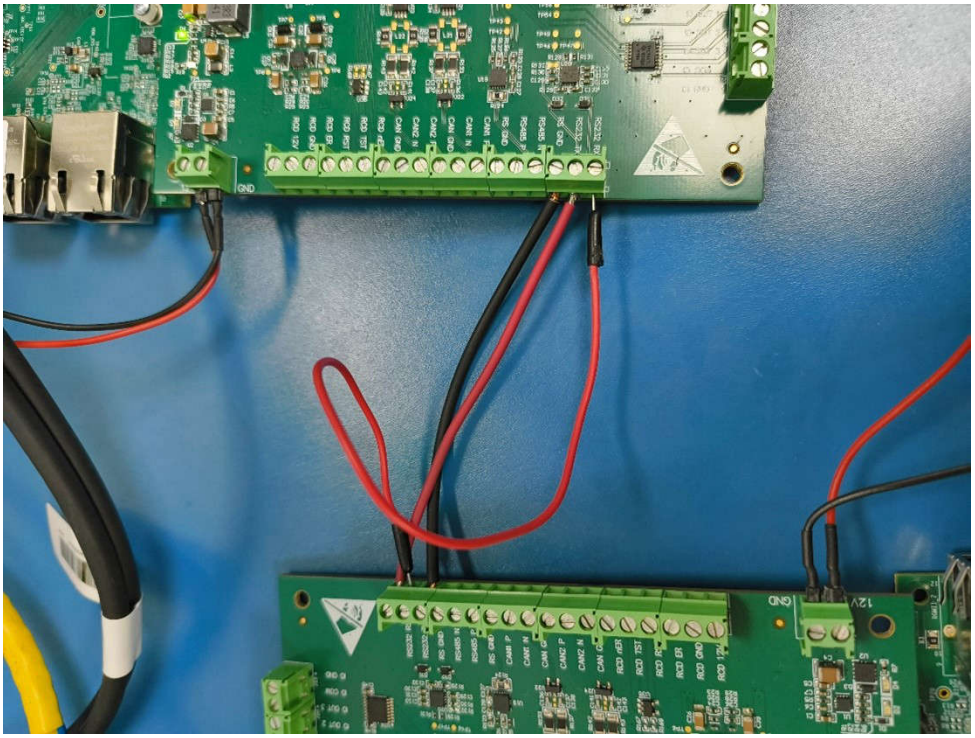


图 8-3. RS232 测试

如前文所述，有一个器件树源叠加层 (dtso) 文件描述了 AM62L-EVSE-DEV-EVM 电路板上的接口均以器件树二进制 (dtbo) 形式提供。AM62L-EVSE-DEV-EVM 电路板的 dtbo 具有必要的 UART 接口配置。代码假定 dtbo 已添加到 U-Boot 中的 uEnv.txt 文件中。

测试：

对于 RS485，Linux 上的 UART 端口为 ttyS4

对于 RS232，Linux 上的 UART 端口为 ttyS3

根据测试，将下面给出的命令中的 <name-of-the-port> 替换为相应的 UART 端口。

使用 stty 命令来配置每个电路板上的 UART 接口

stty -F /dev/<name-of-th-port> 9600 cs8 -cstopb -parenb raw -echo

使用以下命令进行测试

在电路板 A 上：

head -c 34 < /dev/<name-of-th-port>; echo -n "0123456789ABCDEFGHIJKMNOPQRSTUVWXZ" > /dev/<name-of-th-port>

在电路板 B 上：

echo -n "0123456789ABCDEFGHIJKMNOPQRSTUVWXZ" > /dev/<name-of-th-port>; head -c 34 < /dev/<name-of-th-port>

您应该会看到两个电路板上都在发送和读取数据。请参见下图：

电路板 A：

```
root@am62lxx-evm:~# head -c 34 < /dev/ttyS3; echo -n "0123456789ABCDEFGHIJKMNOPQRS
TUVWXZ" > "/dev/ttyS3"
0123456789ABCDEFGHIJKMNOPQRSTUVWXZroot@am62lxx-evm:~#
```

电路板 B：

```
root@am62lxx-evm:~# echo -n "0123456789ABCDEFGHIJKMNOPQRSTUVWXZ" > "/dev/ttyS3"; h
ead -c 34 < /dev/ttyS3
0123456789ABCDEFGHIJKMNOPQRSTUVWXZroot@am62lxx-evm:~#
```

如果发送和接收的数据不相同，则意味着测试失败。

9 MSPM0 的实验室测试设置

此测试通过在 AM62L-EVSE-DEV-EVM 上使用 EV 仿真电路，根据 IEC 61851-1 标准模拟交流充电会话，并解释了 EVerest 的基本配置。

硬件设置

- 将 AM62L-EVM 连接至 AM62L-EVSE-DEV-EVM。
- 将 12V 电源连接到 AM62L-EVSE-DEV-EVM 上的螺钉接线端子 J1。
- 在 AM62L-EVSE-DEV-EVM 与测试 PC 之间连接 USB-C 线缆。
- 在 AM62L-EVM J7 与测试 PC 之间连接 USB-Micro 线缆，以实现串行通信。
- 将 USB-C 电源连接至 AM62L Header J17。
- 在 J5 上的引脚 1 和引脚 2 (EVSE) 之间连接一根跳线，以将 PWM 输出连接到螺钉接线端子 J4。
- 在螺钉接线端子 J4 引脚 3 (CCS CP) 和引脚接头 J5 引脚 3 之间连接一根跳线。此导线用于将控制引导信号连接到 EV 仿真电路。
- 将开关 S1 向右并沿引脚 3 的方向设置，以暂时断开控制引导信号与 EV 仿真电路的连接。
- 将跳线连接到螺钉接线端子 J21 引脚 2 (IO IN 1)。该跳线稍后用于启用充电状态 C，即 EV 通过将其连接到逻辑高电平来请求电源，例如 J2 引脚 2 (5V)。
- 在螺钉接线端子 J4 引脚 1 (PP) 和螺钉接线端子 J4 引脚 2 (PE) 之间连接一个电阻器，以指示已连接充电电缆。电阻器的值将指示充电电缆的最大充电电流能力。1.5kΩ 电阻器可支持高达 13A 的电流。必须设置跳线 J6。
- 可选择将示波器探头连接到控制引导信号 (TP19)。
- PLC 需要在上电后复位。因此，AM62L-EVSE-DEV-EVM 必须在 AM62L 之前通电。

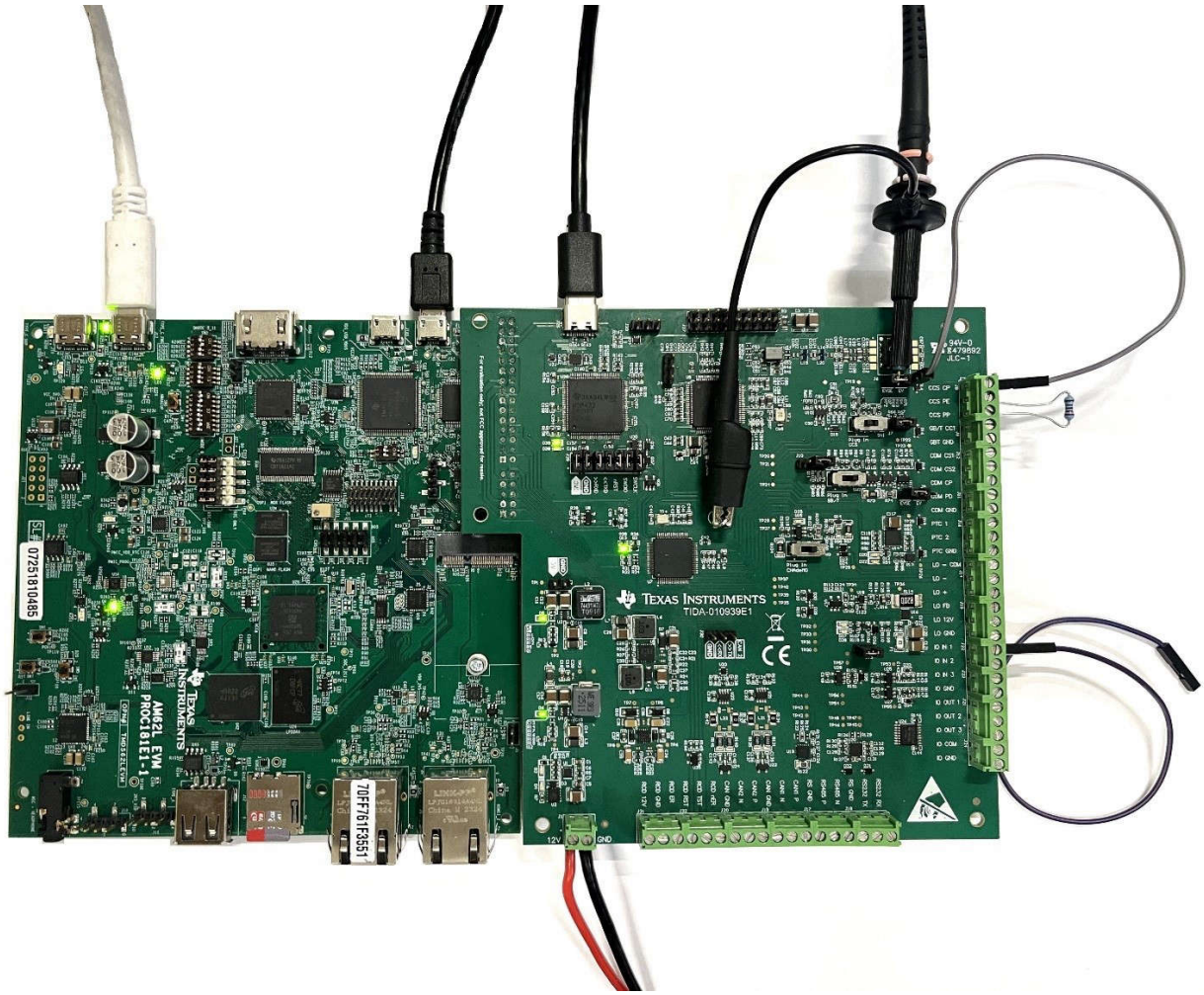


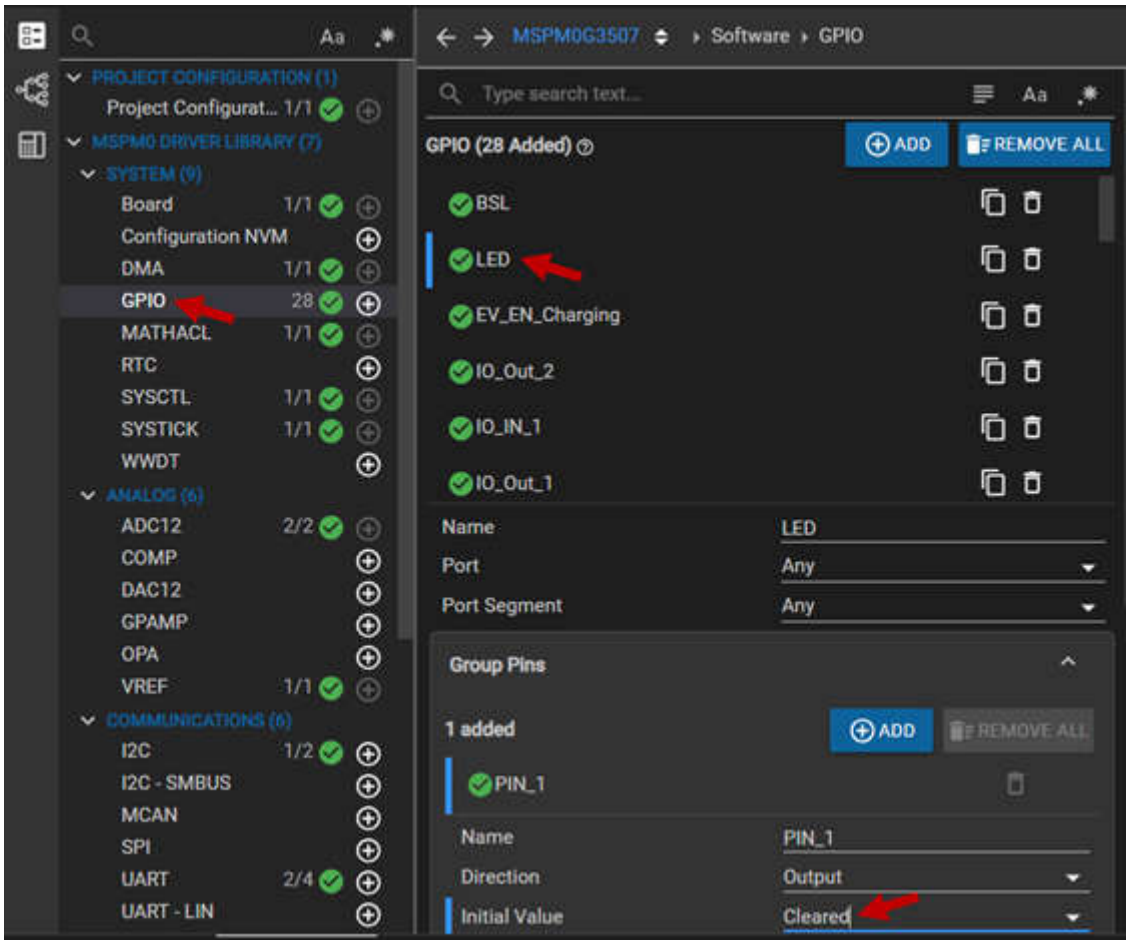
图 9-1. 硬件设置

MSPM0 软件设置

打开 Code Composer Studio 并选择 Browse software and examples，然后导入位于 Examples - LP-MSPM0G3507 - Demos 下的 evse_controller_TIDA-010939 工程。这个示例工程是为了与连接的 TIDA-010239 交流充电器一起工作。因此，如果 AM62L-EVSE-DEV-EVM 独立使用，则必须更改代码。

允许进行电力传输时，继电器必须靠近以将 EV 连接到电网。在此示例中，LED 将指示电源传输何时处于活动状态。在 MSPM0 SDK 软件示例中，LED 默认情况下一直处于活动状态，必须相应地进行更改：

- 打开 Code Composer Studio 并选择 Browse software and examples
- 导入位于 MSPM0 SDK 中 Examples / Demos 下的 evse_controller_TIDA-010939 工程。
- 打开 TIDA-010939.syscfg，在菜单中打开 GPIO，然后选择名为 LED 的 GPIO。
- 在 Group Pins 下，可以将启动后的引脚状态设置为 Initial Value。由于此 LED 应仅处于活动状态，因此当电源传输正在进行时，Initial Value 将设置为 Cleared。



- 之后，关闭并保存 Sysconfig，然后打开位于 modules 文件夹中的 PowerSwitch.cpp。PowerSwitch.cpp 负责处理高压继电器的控制。在 PowerSwitch.cpp 中，可以找到以下函数：PowerSwitch::switchOn() 和 PowerSwitch::switchOff()。

这些函数将启用/禁用 PWM，作为继电器驱动器的控制信号。切换继电器后，MSPM0 将读取反馈信号（也称为焊接检测），以验证继电器的状态。如果继电器切换成功，表明触点未焊接在一起，则变量 relaysHealthy 将设置为 true。由于焊接检测硬件在 TIDA-010939 上不可用，因此 relaysHealthy 必须在代码中以硬编码方式设定。

- 添加到 PowerSwitch::switchOn() 函数的末尾：

```
DL_GPIO_setPins(LED_PORT, LED_PIN_1_PIN);        // enables LED
relaysHealthy = true;                 // overwrites weld detection
```

- 添加到 PowerSwitch::switchOff() 函数的末尾：

```
DL_GPIO_clearPins(LED_PORT, LED_PIN_1_PIN);        // disables LED
relaysHealthy = true;                 // overwrites weld detection
```

```
bool PowerSwitch::switchOn()
{
    if (relaysHealthy) {
        DL_Timer_startCounter(pwmTimer);
        setPWM1(100);
        setPWM2(100);
#ifdef FINE_GRAIN_DEBUG_PRINTF
        printf("switchOn 1: On");
#endif
        //delay_ms(relaysDelay);
        delay_cycles(relaysDelayCycles);
        relaysOn = true;
        setPWM1(relaysHoldingPercent);
        setPWM2(relaysHoldingPercent);
#ifdef FINE_GRAIN_DEBUG_PRINTF
        printf("switchOn 2: PWM mode");
#endif
        if (relayStatus.read())
            relaysHealthy = true;
        else
            relaysHealthy = false;
    }
    DL_GPIO_setPins(LED_PORT, LED_PIN_1_PIN);
    relaysHealthy = true;
    return relaysHealthy;
}
```

```
bool PowerSwitch::switchOff()
{
    //setPWM1(0);
    //setPWM2(0);
    DL_Timer_stopCounter(pwmTimer);
#ifdef FINE_GRAIN_DEBUG_PRINTF
    printf("switchOn 1: Off");
#endif
    //delay_ms(relaysDelay);
    delay_cycles(relaysDelayCycles);
#ifdef FINE_GRAIN_DEBUG_PRINTF
    printf("switchOn 2: Off after delay");
#endif
    relaysOn = false;
    if (!relayStatus.read())
        relaysHealthy = true;
    else
        relaysHealthy = false;
    DL_GPIO_clearPins(LED_PORT, LED_PIN_1_PIN);
    relaysHealthy = true;
    return relaysHealthy;
}
```

在此示例中，使用 AM62L-EVSE-DEV-EVM 中包含的 EV 仿真电路来仿真 EV 的行为。开关 SW1 通过将一个 2.74kΩ 电阻器放置到控制引导信号来连接 EV – 状态 B (9V)。状态 C (6V) – 通过将附加的 1.30kΩ 电阻连接到 CP 信号，可以请求能量传输。该电阻器由 MSPM0 通过 MOSFET 进行控制。在此测试中，状态 C 应由 AM62L-EVSE-DEV-EVM 的 IO IN 1 通过连接到它的跳线手动控制。如果输入设置为逻辑高电平，则会启用状态 C 以请求电源传输。因此，必须读出 IO IN 1 引脚。

- 添加到 ControlPilot::run_state_machine() 函数的开头，位于 ControlPilot.cpp 中：

```
if (DL_GPIO_readPins(IO_IN_1_PORT, IO_IN_1_PIN_5_PIN) == 1) {
        DL_GPIO_setPins(EV_Charge_PORT, EV_Charge_PIN_3_PIN);
}
else {DL_GPIO_clearPins(EV_Charge_PORT, EV_Charge_PIN_3_PIN);}
```

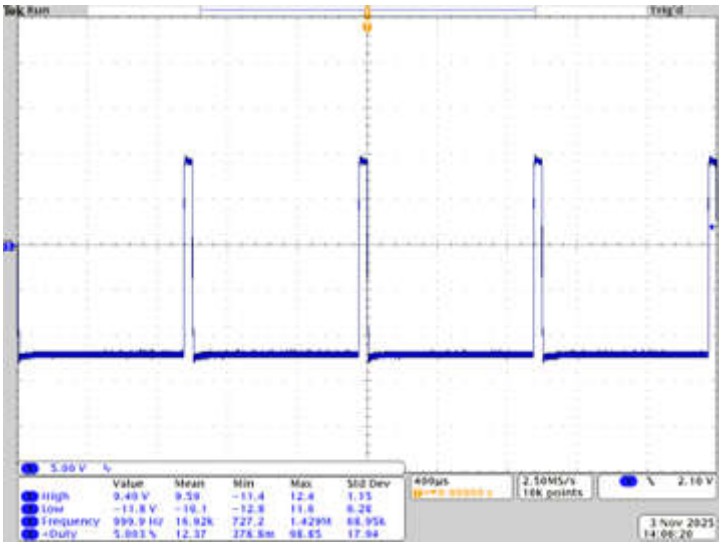
```
void ControlPilot::run_state_machine()
{
    // Sets EV-simulation to state C if IO IN 1 is high
    if (DL_GPIO_readPins(IO_IN_1_PORT, IO_IN_1_PIN_5_PIN) == 1) {
        DL_GPIO_setPins(EV_Charge_PORT, EV_Charge_PIN_3_PIN);
    }
    else { DL_GPIO_clearPins(EV_Charge_PORT, EV_Charge_PIN_3_PIN); }
```

现在，可以在 EVSE 测量控制引导之前更改 EV 的状态。CP 会在检查 RCD、Lock、Proximity Pilot 和启用电源传输的权限后进行测量。Control Pilot 的状态由稍后在 ControlPilot.cpp 中的 ControlPilot::read_from_car 函数设置。

- 编译代码并烧录至 MSPM0。

在 AM62L 上使用 EVerest 进行 EV 充电会话仿真

- 系统启动
  - 将 AM62L EVM 连接至 USB-C 电源。
  - 在 PC 上打开串行终端（COM 端口）。默认波特率为 115200。
- 系统引导后，登录到 Linux shell。
- 要实时监视 EVerest 日志，请运行：journalctl -fu everest
- 将示波器探头连接到 CCS 控制引导 (CP) 信号 (TP19)
  - 空闲电压电平应约为 +12V
- 在 EVerest 日志中，将看到类似如下的消息：
  - 电流限制：6
  - 准备开始充电
- 通过将开关 1 (SW1) 移动到 Plug-IN 位置（朝向引脚 1）来连接 EV 仿真电路。
- 现在，CP 电压在 +9V / −12V 之间切换，对应于状态 B – EV 已连接。
- PWM 占空比为 5%，用于发出数字通信请求 (HLC / ISO 15118)。在此配置中，EVerest 最初尝试使用 ISO 15118 (HLC) 开始充电：
  - CP 状态已更改： A -> B
  - EVSE IEC 会话已启动： EVConnected
  - EVSE IEC Set PWM On (5.0%) 耗时 0ms



- EVerest 等待大约 30 秒，以接收来自 EV 的 PLC（电力线通信）响应：
  - EVSE IEC 交流模式，启用 HLC (ac_enforce_hlc)，在发出 dlink 错误信号之前保持 5% 导通状态
- 如果未建立有效的 HLC 连接，则 PWM 将复位并启动旧的 IEC 61851-1 会话：
  - EVSE IEC EV 未转换到状态 C，因此尝试根据 IEC61851-1 A.5.3 进行一次旧唤醒
- 在经历第二次 30 秒的 HLC 超时之后，EVerest 启动基于 IEC 61851-1 PWM 的传统充电会话：
  - EVSE IEC Set PWM On (10%) 耗时 0ms
- 现在 PWM 设置为 10%，这意味着允许 EV 汲取最高 6A 的电流。此 EVerest 设置中，这等于配置的充电器的最大电流能力。现在，可通过将 IO IN 1 连接到 5V（例如 J2 引脚 2）来请求电源传输（状态 C）。日志将显示更新后的状态和充电过程的开始：
  - CP 状态已更改： B -> C
  - CAR IEC 事件： CarRequestedPower

  - EVSE IEC 充电器状态：CarPaused -> 正在充电
- AM62L-EVSE-DEV-EVM 电路板上的 LED D13 将亮起，表示继电器已关闭，向 EV 的电力传输处于活动状态。如果跳线与 IO IN 1 断开连接，充电会话将暂停，直到 EV 恢复到状态 C，或者通过断开 EV 与充电桩的连接来结束充电会话。

更改 EVerest 配置

在默认设置中，EVerest 根据 ISO15118 配置为交流充电，单相允许最大充电电流为 6A。EVerest 配置存储在 AM62L 上，具体位置如下：

/etc/everest/config.yaml

配置文件 config.yaml 定义了哪些模块处于活动状态、模块的连接方式以及每个模块的参数是什么。每个模块都对应于 EVSE 中的逻辑功能，例如身份验证、计量或通信。

表 9-1. 模块和函数

模块名称	功能
evse_manager	主充电点逻辑：用于管理插头状态、电流限制、PWM 生成和会话处理。
grid_connection_point	用于定义电网限制（保险丝尺寸和可用相位等）。充当上游能量节点。
tida_mcu	AM62L-EVSE-DEV-EVM 电路板的硬件接口。用于控制接触器、CP 信号、RCD、lock 和读取输入。
auth	用于管理用户授权（RFID、即插即充或外部访问）。
api	提供用于监控和控制的 REST/WebSocket 接口。由仪表板或外部系统使用。
energy_manager	在多个 EVSE 或电网节点之间执行负载平衡。
persistent_store	将会话和配置数据存储在数据库 (SQLite) 中。
token_provider / token_validator	用于测试或集成外部身份验证后端 (RFID、OCPP)。
Slac	用于处理 HomePlug GreenPHY 通信（对于 ISO 15118 是必需的）。

在此配置中，ISO15118 由 EvseV2G 模块表示，位于 iso15118_charger 模块中：

iso15118_charger：
- module: EvseV2G
- config_module:
  - device: seth0 # PLC network interface
- connections:
  - security:
    - implementation_id: main
    - module_id: evse_security

在 evse_manager 中设置数字通信 (HLC)：
- ac_hlc_enabled: true
- ac_enforce_hlc: true

如果禁用，EVerest 将回退到基于 IEC 61851-1 PWM 的模拟充电模式。

若要更改相位数和最大电流设置，必须调整 grid_connection_point 和 tida_mcu 模块。以下示例显示了一个三相 16A 配置：
- 电网连接点：用于定义可用的电网容量，对于负载共享很重要：
  - grid_connection_point:
  - config_module:
    - fuse_limit_A: 16
    - phase_count: 3
- MCU 接口（TIDA 电路板）：用于定义电流和相位数的硬件级限制：

- tida_mcu:
- module: TIDA010939BSP
- config_module:
  - serial_port: /dev/ttyS2
  - baud_rate: 115200
  - min_current_A_import: 6
  - max_current_A_import: 16
  - min_phase_count_import: 3
  - max_phase_count_import: 3
  - has_socket: true

将其设置为 16A 会将占空比更改为 ≈27%，对应于每相 16A。有关 EVerest 的更多信息，请访问 everest.github.io。

10 总结

AM62L-EVSE-DEV-EVM 用户指南提供了设置和测试电动汽车供电设备 (EVSE) 开发平台的全面说明，该平台将 AM62L 处理器与 AM62L-EVSE-DEV-EVM 参考设计相结合 [1]。本文档首先介绍了系统架构，该架构采用搭载 EVerest 开源充电栈的 AM62L 处理器来实现数字通信，并采用 MSPM0G3507 微控制器处理模拟握手和安全功能。

本指南详细介绍了使用两个 AM62L EVM 和 AM62L-EVSE-DEV-EVM 电路板的测试设置建议，无需昂贵的测试设备，即可仿真 EVSE 到 EV 通信。其中介绍了包括器件树叠加层在内的软件配置，并说明了如何测试三个关键通信接口：

1. 使用 Lumissil Green PHY 器件进行电力线通信 (PLC) 测试，用于 EVSE 和 PEV 之间的 IP 网络
2. 用于功率转换单元控制的 CAN 总线通信
3. 通过 RS-485 和 RS-232 接口进行的 UART 通信

此外，本文档还提供了 MSPM0 微控制器的详细实验室测试设置，根据 IEC 61851-1 标准来仿真交流充电会话。其中介绍了如何修改 Code Composer Studio 中的 MSPM0 代码，以及如何配置 EVerest 软件栈来控制最大电流和相位数等充电参数。

本指南是开发人员构建 EV 充电解决方案的综合资源，使开发人员能够根据不同的充电标准和地区规定，测试和验证现代 EVSE 系统中所需的各种通信接口。

11 其他信息和资源

- AM62L 产品页面
- AM62L 产品概述
- AM62L 器件学院
- AM62L-EVSE-DEV-EVM 工具文件夹
- AM62L-LINUX-SDK
- 电动汽车供电设备前端控制器参考设计（修订版 A）
- AM62L-EVSE-DEV-EVM 用户指南
- MSPM0G3507 产品页面

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月