

# Getting started with AEK-POW-PDUMINI - 12 V power distribution unit (PDU) board with 10 output channels, of which 8 paired with P-channels for superb standby performance

## Introduction

In recent years, the power distribution unit (PDU) market has been substantially revamped by the increasing adoption of electrical vehicles (EVs). The number of supplies in the vehicle has been increasing, while the need to reduce vehicle harness for better battery usage has pushed a strong evolution of the power distribution that remains a key function of the E/E architecture.

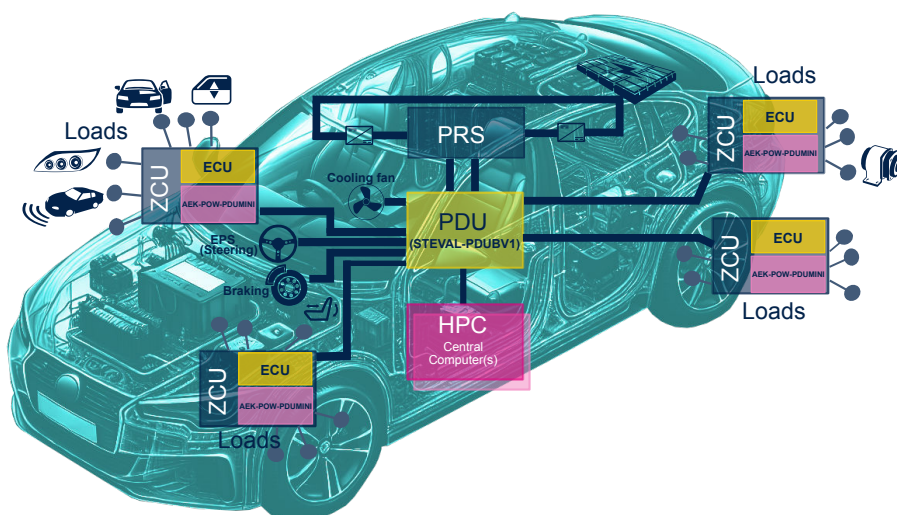
Automotive power distribution systems handle the transport of low-voltage electric energy via the wire harness. A primary PDU enables the connections with loads and supply in a single location in the vehicle. Designed as an all-in-one single unit to optimize space, it allows the protection of the primary circuits (preheating, electric steering, body computer, etc.) and the distribution of the engine or battery power to various electrical components, including the motor, power electronics, and auxiliary systems (infotainment, lights, mirrors, etc.).

The key trends in the global electric PDU market include the adoption of 12 V and 48 V bus systems, the adoption of e-fuses to replace traditional fuse boxes, and the transfer of secondary power delivery functions to the zone, building the new vehicle E/E architecture.

The modern power distribution system may consist of several blocks: the power rail switch unit (PRS) having multiple power source connections (for example, main battery and auxiliary battery), the primary PDU directly accessing the PRS power sources, and distributing the power throughout the vehicle by several e-fuse channels. Some channels are for direct supply of motors and loads, while other channels supply a secondary/zonal power distribution unit (for example, ADAS function distribution domain or rear zone power distribution unit).

**AEK-POW-PDUMINI** perfectly fits into this market trends. It can be used as a secondary power distribution unit that is closer to the loads inside the zone control unit, reducing the wire total length. This integration of the secondary power distribution unit in the zone control unit optimizes the power distribution wire harness, and the in-vehicle network (IVN) too. The same **AEK-POW-PDUMINI** can also be used as a 'building block' for complex primary PDUs.

**Figure 1. Example of a possible vehicle power distribution system**



**Note:** The evaluation board described in this user manual is designed for R&D laboratory use only. It is not intended for field use in vehicles. Moreover, it is not a reference design. Its purpose is evaluation and NOT production, as stated in our [Terms of use](#).

**Notice:** For dedicated assistance, submit a request through our online support portal at <https://community.st.com/autodevkit>.

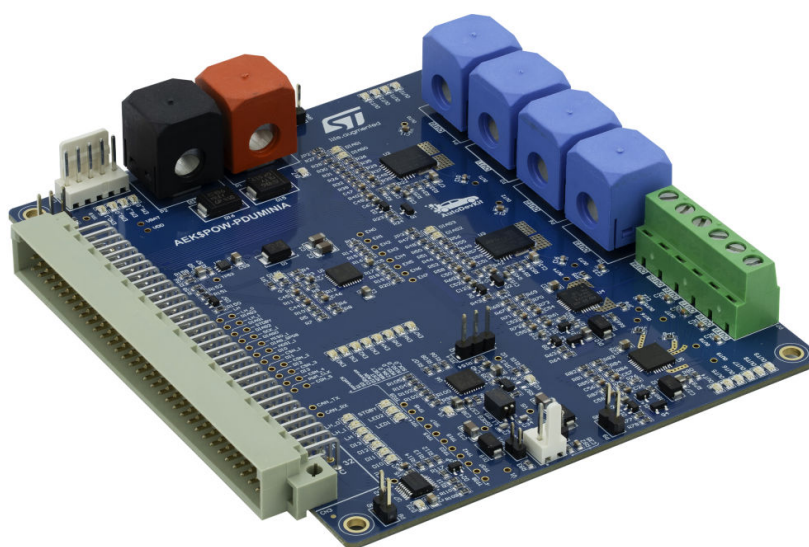
## 1 Overview

The **AEK-POW-PDUMINI** board is based on last-generation programmable eFuses and can be easily connected to the **AEK-MCU-C4MLIT3** MCU board (or the soon available **AEK-MCU-C4MLIT2**) using a standardized 96-pin plug-and-play automotive unified connector. Other automotive MCUs like Stellar and STM32A will be supported in the future using the same connector. For example, **AEK-MCU-C4MLIT2** plus **AEK-POW-PDUMINI** enables power delivery control through several protocols like 10BASE-T1S, 100BASE-T1X, and CAN-FD.

The **AEK-POW-PDUMINI** board is intended for bench/laboratory use at standard room temperature range.

Finally, the advantage of using **AEK-POW-PDUMINI** is that it is a functional block of the AutoDevKit ecosystem. This makes the board easy to configure and to use, thanks to dedicated demos available in **AutoDevKit Studio**.

**Figure 2. AEK-POW-PDUMINI top side view**



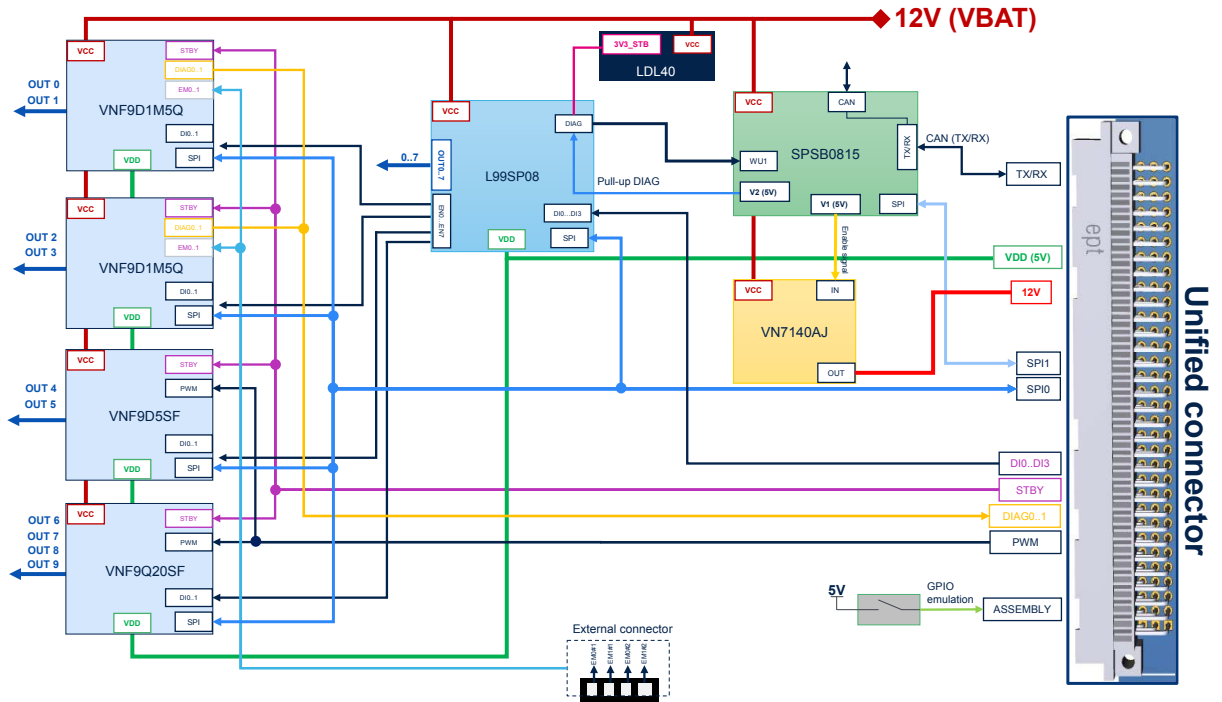
### 1.1 Deliverable package contents

The package includes an **AEK-POW-PDUMINI** board.

### 1.2 Supported key devices

The key supported devices are:

1. **L99SP08**: smart octal P-channel for ultralow current consumption in automotive systems, featuring 400 mA per channel, able to interact with the high-side drivers with STi<sup>2</sup>Fuse. It is controllable via SPI from MCU.
2. **VNF9D1M5Q**: 2-channel high-side driver with STi<sup>2</sup>Fuse protection for automotive power distribution applications, featuring 27 A per channel. The board features two instances of this device.
3. **VNF9D5SF**: 2-channel high-side driver with STi<sup>2</sup>Fuse protection for automotive power distribution applications, featuring 15 A per channel.
4. **VNF9Q20SF**: 4-channel high-side driver with STi<sup>2</sup>Fuse protection for automotive power distribution applications, featuring 6 A per channel.
5. **SPSB081**: automotive power management IC (PMIC) with LIN and CAN-FD, which manages the wake-up for the MCU, for example following a Parking mode to Running mode application state transition.
6. **VN7140AJ**: high-side driver with current feedback for automotive applications, used to power the MCU board from 12 V (VBAT) at power up and after a wake up condition.
7. **LDL40**: 200 mA low dropout LDO.

**Figure 3. AEK-POW-PDUMINI conceptual block diagram**


### 1.3

## Technology background on high-side drivers with STi<sup>2</sup>Fuse

The STi<sup>2</sup>Fuse protection (ST proprietary I<sup>2</sup>t functionality) features an intelligent circuit breaking aimed at protecting PCB traces, connectors, and wire harness from overheating, with no impact on load transients such as inrush currents and capacitance charging.

The I<sup>2</sup>t curve parameters can be individually set per each channel. The dissipated power can be limited to a safe level up to thermal shutdown intervention.

Thermal shutdown can be configured as latched off or programmable time-limited auto-restart. The output current limitation protects the device in case of overload.

The device enters a fail-safe mode in case of communication loss with the microcontroller, reset of digital memory, or watchdog monitoring time-out event. In fail-safe mode, the outputs can be directly controlled by dedicated, assignable direct inputs.

The high-side drivers are activated only when the MCU sends them a “STANDBY\_NOT” control signal.

Eight out of ten high-side driver channels present in AEK-POW-PDUMINI are connected to L99SP08 P-channel outputs. The remaining two channels are independent. In Standby mode (for example, car in Park mode), the eight main output channels are switched off while the coupled L99SP08 P-channels are turned on. As soon as the power demand is greater than the programmable threshold, the process is inverted, that is, the eight main output channels are turned on while the P-channels are switched off. This behavior is applicable to any of the eight channels independently from the others. All high-side drivers have an integrated smart fuse with a current threshold configurable via SPI to block the channel current drain at a given certain level. This is towards the trend to replace existing discrete fuses with silicon-based regenerable, configurable, and precise smart fuses.

#### 1.4 **Reverse polarity**

The board is protected against reverse polarity.

#### 1.5 **Shortcut to ground**

No known issues.

#### 1.6 **ISO pulses**

The devices on [AEK-POW-PDUMINI](#) are protected against ISO pulses. However, they are not fully tested.

#### 1.7 **ESD**

The board is equipped with measures against ESD pulses. Communication lines are protected by specialized ESD devices. However, normal ESD handling procedures are recommended.

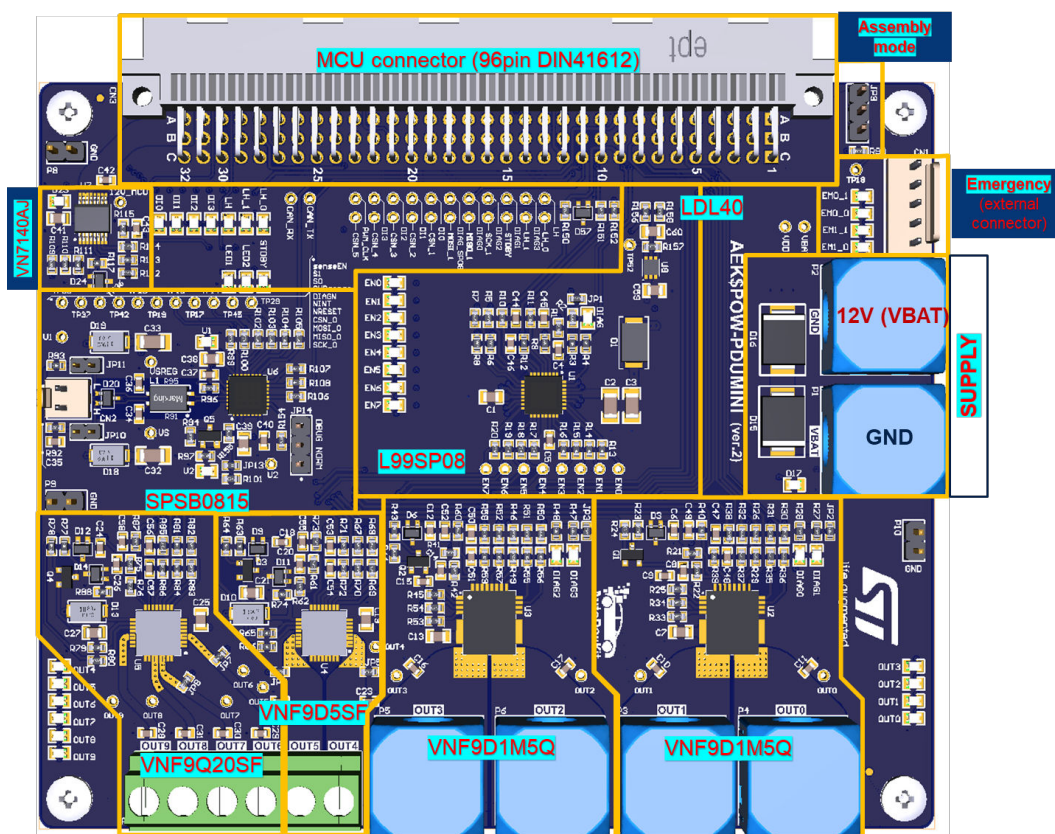


## 2 Hardware description

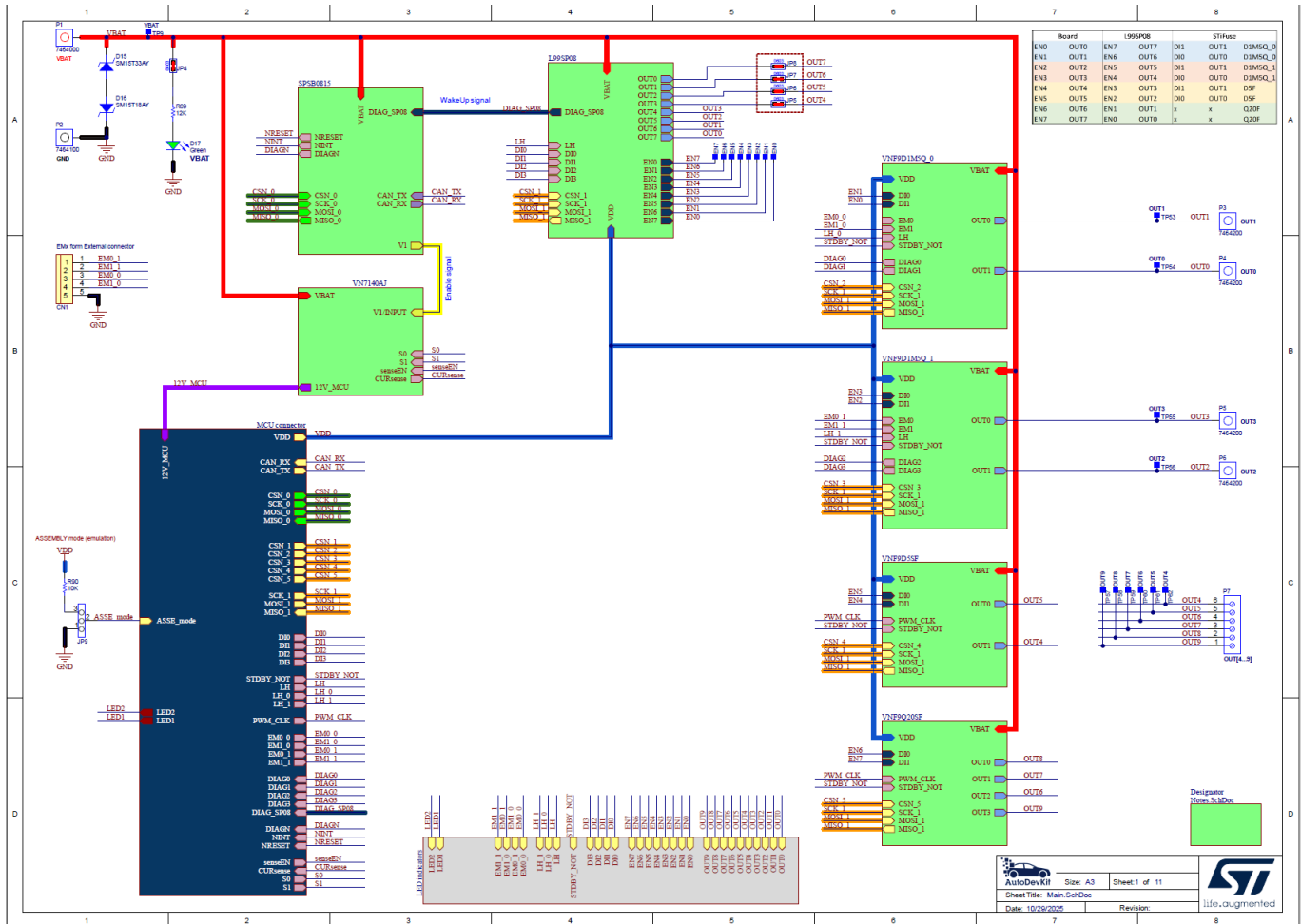
### 2.1 Hardware features

The image below shows the AEK-POW-PDUMINI board layout, highlighting the main functional blocks.

Figure 4. AEK-POW-PDUMINI functional blocks



**Figure 5. AEK-POW-PDUMINI - schematic view**



Each functional block is briefly described below:

- **MCU connector (96-pin DIN41612):** automotive unified connector at 96-pin DIN41612 dedicated to the MCU interface.
- **L99SP08:** smart octal P-channel for ultralow current consumption in automotive systems, featuring 400 mA per channel, able to interact with the high-side drivers with STiFuse. It is controlled by the MCU via SPI.
- **SPSB081:** automotive power management IC with LIN and CAN-FD, which manages the wake-up for the MCU (for example, following a Parking mode to Running mode application state transition).
- **VNF9D1M5Q:** 2-channel high-side driver with STiFuse protection for automotive power distribution applications, featuring 27 A per channel.
- **VNF9D5SF:** 2 channel high-side driver with STiFuse protection for automotive power distribution applications, featuring 15 A per channel.
- **VNF9Q20SF:** 4 channel high-side driver with STiFuse protection for automotive power distribution applications, featuring 6 A per channel.
- **LDL40:** 200 mA low dropout LDO.
- **VN7140AJ:** high-side driver with current feedback for automotive applications, used to power the MCU board from 12 V (VBAT) at power-up and after a wake-up condition.
- **Emergency (ext. connector):** for sending emergency signal for rapid switch-off of the output channels.
- **Assembly mode:** jumper selector for accessing this operating mode.
- **Supply:** power input connectors to supply the board at operating voltage (14 V).

## 2.2 Hardware dimensions

The [AEK-POW-PDUMINI](#) evaluation board has the following dimensions and characteristics:

- PCB area: 120 mm x 100 mm.
- PCB thickness: 1.6 mm.
- 4 layers: top, power1, power2, bottom.
- Copper weight: 2 oz (each layer).

## 2.3 Main connector

The main [AEK-POW-PDUMINI](#) board connector is a 96-pin DIN41612 automotive unified connector dedicated to the MCU. This connector allows to connect the board to other MCU boards, such as the [AEK-MCU-C4MLIT3](#) with remote PDU access via CAN-FD, or the soon available [AEK-MCU-C4MLIT2](#), which can implement additional remote PDU access via 100BASE-TX or 10BASE-T1S.

Pins are split into three rows (A, B, C), each one featuring 32 pins.

The 96-pin connector performs two main functions: it supplies 12V\_MCU power to the MCU board and manages communication with the STi<sup>2</sup>Fuse, the smart octal P-channel, and the PMIC present on [AEK-POW-PDUMINI](#). Furthermore, this connector provides the VDD voltage necessary for the MCU and the digital peripherals of the devices on the [AEK-POW-PDUMINI](#) board to share the same voltage reference.

The 12V\_MCU voltage is supplied via the [VN7140AJ](#) high-side driver controlled by the PMIC at power-up.

The additional signals transmitted through this connector are:

- SPI communication to control the PMIC, STi<sup>2</sup>Fuse devices, and smart octal P-channel.
- CAN communication through the PMIC.
- DIAGx signals to an MCU external interrupt line (diagnostic function).
- Dix/EMx from MCU GPIOs for channel direct control without using SPI (for example, startup, fault management, etc.) and emergency signal for rapid switch-off of the output channels.
- LHx limp home from MCU GPIOs to switch the device from Normal mode to fail-safe mode.
- PWM from an MCU timer to control the outputs in PWM mode.
- STANDBY\_NOT from MCU GPIOs to force the device into Standby mode.
- ASSE\_mode from MCU GPIOs to emulate the Assembly mode.

Figure 6. 96-pin automotive unified connector - board view

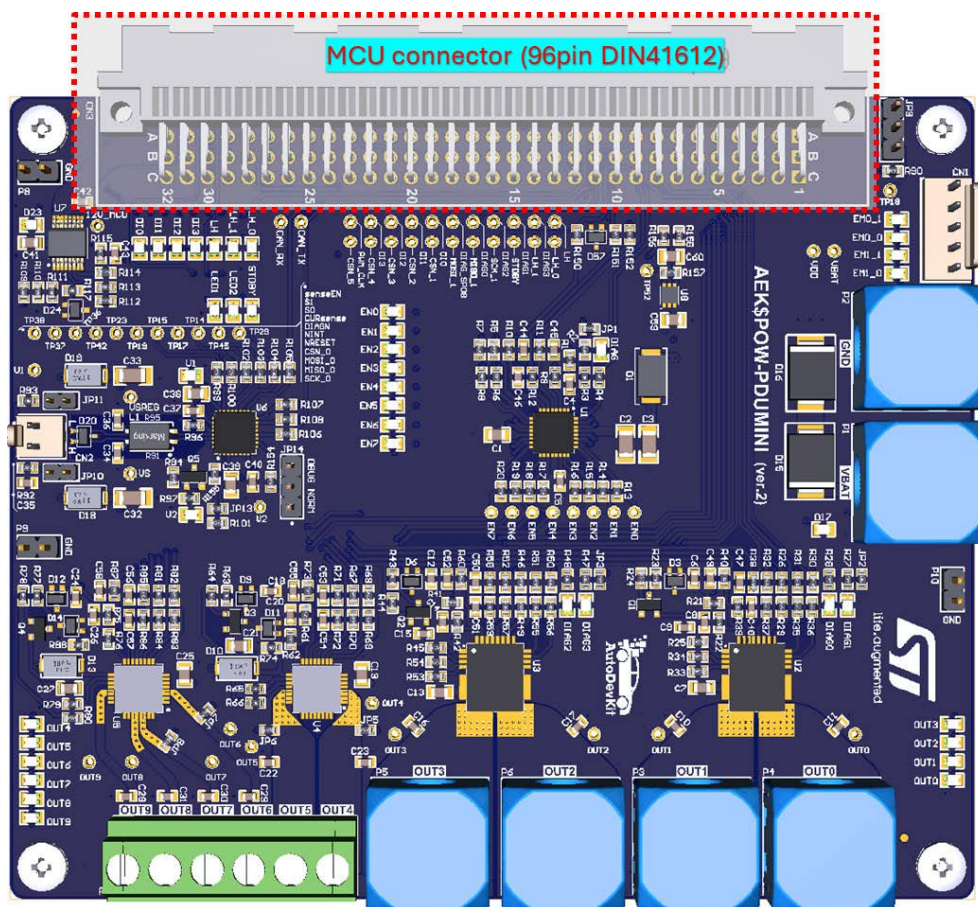


Figure 7. 96-pin automotive unified connector - schematic view

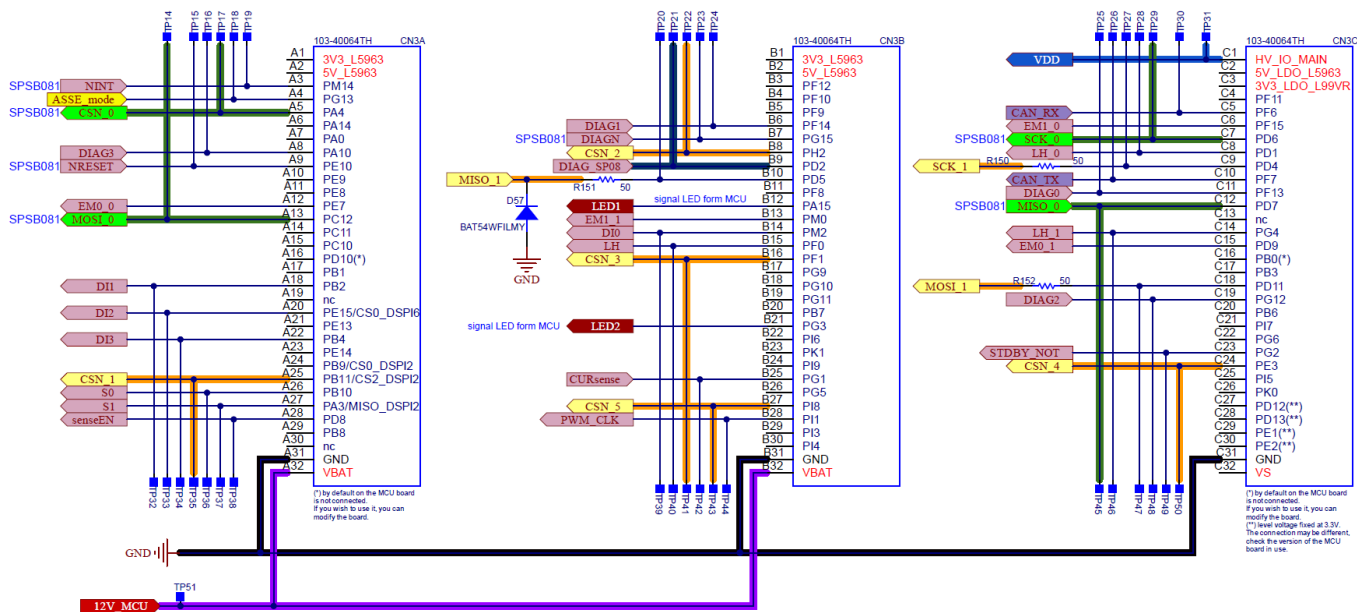


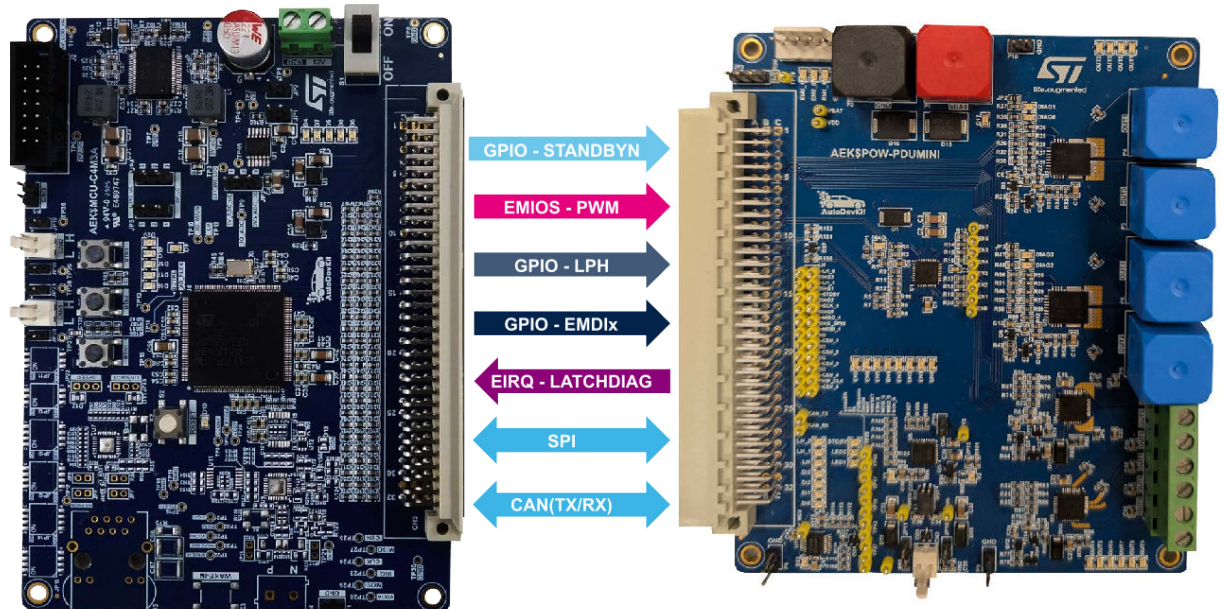


Figure 8. AEK-POW-PDUMINI main MCU connector pinout

| Row A |         |               | Row B |         |               | Row C |         |                                  |
|-------|---------|---------------|-------|---------|---------------|-------|---------|----------------------------------|
| A1    | Power   | 3V3 DC-DC     | B1    | Power   | 3V3 DC-DC     | C1    | Power   | HVIO main (MCU supply)           |
| A2    | Power   | 5V DC-DC      | B2    | Power   | 5V DC-DC      | C2    | Power   | 5V LDO                           |
| A3    | Digital |               | B3    | N.C.    | Not connected | C3    | Power   | 3V LDO                           |
| A4    | Digital |               | B4    | N.C.    | Not connected | C4    | N.C.    | Not connected                    |
| A5    | Digital |               | B5    | N.C.    | Not connected | C5    | Digital |                                  |
| A6    | N.C.    | Not connected | B6    | Digital |               | C6    | Digital |                                  |
| A7    | N.C.    | Not connected | B7    | Digital |               | C7    | Digital |                                  |
| A8    | Digital |               | B8    | Digital |               | C8    | Digital |                                  |
| A9    | Digital |               | B9    | Digital |               | C9    | Digital |                                  |
| A10   | N.C.    | Not connected | B10   | Digital |               | C10   | Digital |                                  |
| A11   | N.C.    | Not connected | B11   | N.C.    | Not connected | C11   | Digital |                                  |
| A12   | Digital |               | B12   | Digital |               | C12   | Digital |                                  |
| A13   | Digital |               | B13   | Digital |               | C13   | N.C.    | Not connected                    |
| A14   | N.C.    | Not connected | B14   | Digital |               | C14   | Digital |                                  |
| A15   | N.C.    | Not connected | B15   | Digital |               | C15   | Digital |                                  |
| A16   | N.C.    | Not connected | B16   | Digital |               | C16   | N.C.    | Not connected                    |
| A17   | N.C.    | Not connected | B17   | N.C.    | Not connected | C17   | N.C.    | Not connected                    |
| A18   | Digital |               | B18   | N.C.    | Not connected | C18   | Digital |                                  |
| A19   | N.C.    | Not connected | B19   | N.C.    | Not connected | C19   | N.C.    | Not connected                    |
| A20   | N.C.    | Not connected | B20   | N.C.    | Not connected | C20   | N.C.    | Not connected                    |
| A21   | Digital |               | B21   | Digital |               | C21   | N.C.    | Not connected                    |
| A22   | Digital |               | B22   | N.C.    | Not connected | C22   | N.C.    | Not connected                    |
| A23   | N.C.    | Not connected | B23   | N.C.    | Not connected | C23   | Digital |                                  |
| A24   | N.C.    | Not connected | B24   | N.C.    | Not connected | C24   | Digital |                                  |
| A25   | Digital |               | B25   | Digital |               | C25   | N.C.    | Not connected                    |
| A26   | Digital |               | B26   | N.C.    | Not connected | C26   | N.C.    | Not connected                    |
| A27   | Digital |               | B27   | Digital |               | C27   | N.C.    | Not connected                    |
| A28   | Digital |               | B28   | Digital |               | C28   | N.C.    | Not connected                    |
| A29   | N.C.    | Not connected | B29   | N.C.    | Not connected | C29   | N.C.    | Not connected                    |
| A30   | N.C.    | Not connected | B30   | N.C.    | Not connected | C30   | N.C.    | Not connected                    |
| A31   | Power   | GND           | B31   | Power   | GND           | C31   | Power   | GND                              |
| A32   | Power   | VBAT 12V      | B32   | Power   | VBAT 12V      | C32   | Power   | VS (12 V after reverse polarity) |

The following figure shows an example of AEK-MCU-C4MLIT3 connected to AEK-POW-PDUMINI.

Figure 9. AEK-MCU-C4MLIT3 main MCU board connected to AEK-POW-PDUMINI PDU board



## 2.4 Power input connectors

The board is powered via two press fit connectors (VBAT and GND) that support up to 120 A of nominal current. The nominal operating voltage of the AEK-POW-PDUMINI board is 14.0 V. According to the operating voltages for STi2Fuse devices, the maximum operating voltage is 28 V.

## 2.5 Power output connectors

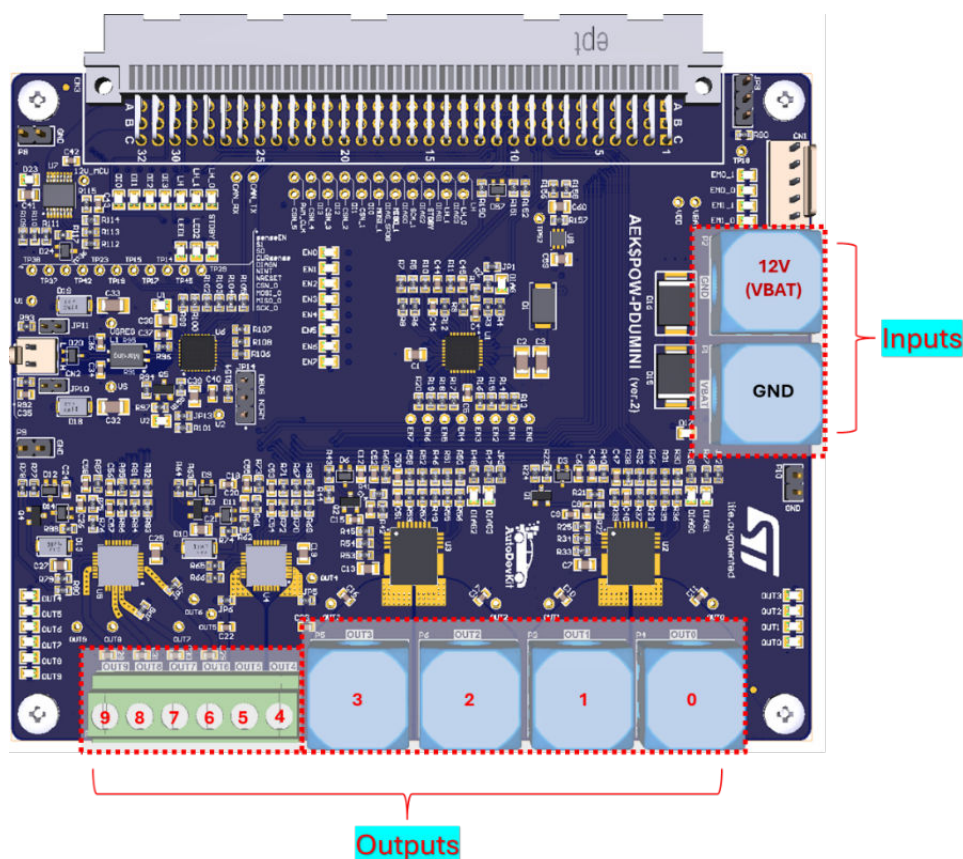
The AEK-POW-PDUMINI power outputs are provided via two types of connectors:

- OUT0, OUT1, OUT2, and OUT3 are of press fit type, supporting higher current capacity.
- OUT4, OUT5, OUT6, OUT7, OUT8, and OUT9 are rising cage clamp type at 24 A (VDE standard).

**Table 1. Output connectors and related output values**

| Channel | Max. current | Device                |
|---------|--------------|-----------------------|
| OUT0    | 27 A         | VNF9D1M5Q and L99SP08 |
| OUT1    | 27 A         |                       |
| OUT2    | 27 A         |                       |
| OUT3    | 27 A         | VNF9D1M5Q and L99SP08 |
| OUT4    | 15 A         | VNF9D5SF and L99SP08  |
| OUT5    | 15 A         |                       |
| OUT6    | 6 A          | VNF9Q20SF and L99SP08 |
| OUT7    | 6 A          |                       |
| OUT8    | 6 A          | VNF9Q20SF             |
| OUT9    | 6 A          | VNF9Q20SF             |

**Figure 10. Power input and output connectors, board view**



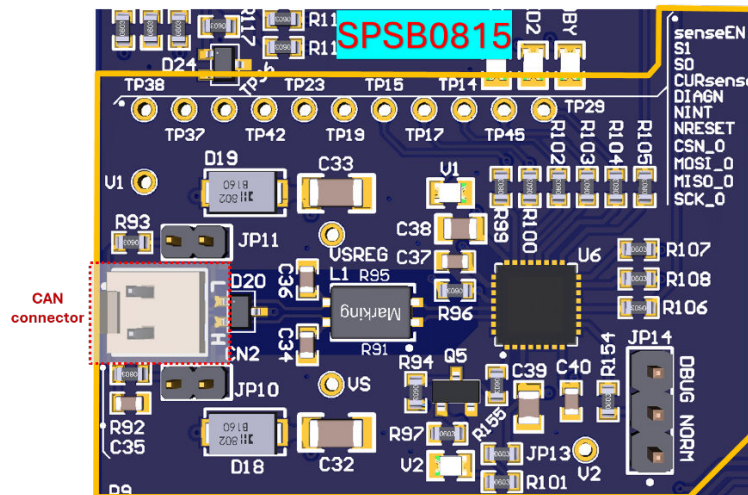


## 2.6 CAN communication connector

The CAN communication interface is used to send commands that control devices and channels on **AEK-POW-PDUMINI**, as well as to support the UDS diagnostic protocol.

The CAN physical layer is embedded in the on-board **SPSB0815** PMIC. This device can wake up the MCU and the entire system when traffic is detected on the CAN bus.

**Figure 11. CAN connector - board view**

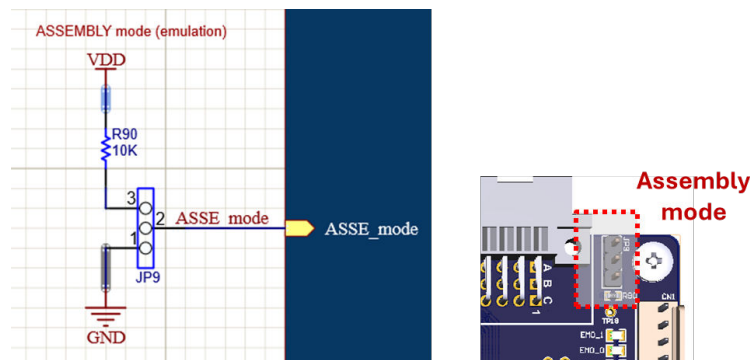


## 2.7 Assembly mode

The Assembly mode is an operating mode of **AEK-POW-PDUMINI** used in the vehicle production line by the car maker to ensure worker safety.

Access to this operating mode is controlled by a jumper selector (JP9) that the MCU detects during the start-up phase. Setting this pin to VDD (2-3, logic level 1) disables all STi<sup>2</sup>Fuse outputs.

**Figure 12. AEK-POW-PDUMINI Assembly mode emulation - schematic and board view**



Refer to [Section 4.2.12.1](#) for further details on Assembly mode.

## 2.8 Parking mode

Parking mode is defined as the state of the vehicle when it stops after Running mode (that is, driving). This mode is simulated through a dedicated CAN message.

During this mode, **AEK-POW-PDUMINI** should focus on reaching the lowest possible power consumption. Thus, the following operations are carried out:

- MCU is switched off.
- All STi<sup>2</sup>Fuse devices enter fail-safe mode.
- PMIC enters Standby mode (that is, the lowest power consumption).

Before entering Parking mode and shutting down, the MCU must perform a thorough check of the current consumption of each channel to ensure that it is below the reactivation threshold for all VNFx devices.

Once Parking mode is entered, the STi<sup>2</sup>Fuse output channels are switched off (through the STDBY\_NOT pin), while the coupled L99SP08 P-channels remain active if they were previously activated by a CAN message.

The L99SP08 P-channel outputs can provide a maximum current up to 400 mA (threshold programmable by SPI). As soon as the power demand exceeds the programmed threshold, the process is inverted, that is, the STi<sup>2</sup>Fuse output channels (with the activated L99SP08 P-channel) are turned on while the P-channels are turned off.

The conditions to exit Parking mode are:

- A current increase request on one of the channels sustained by the L99SP08 P-channels.
- A wake up event due to bus activity on CAN.

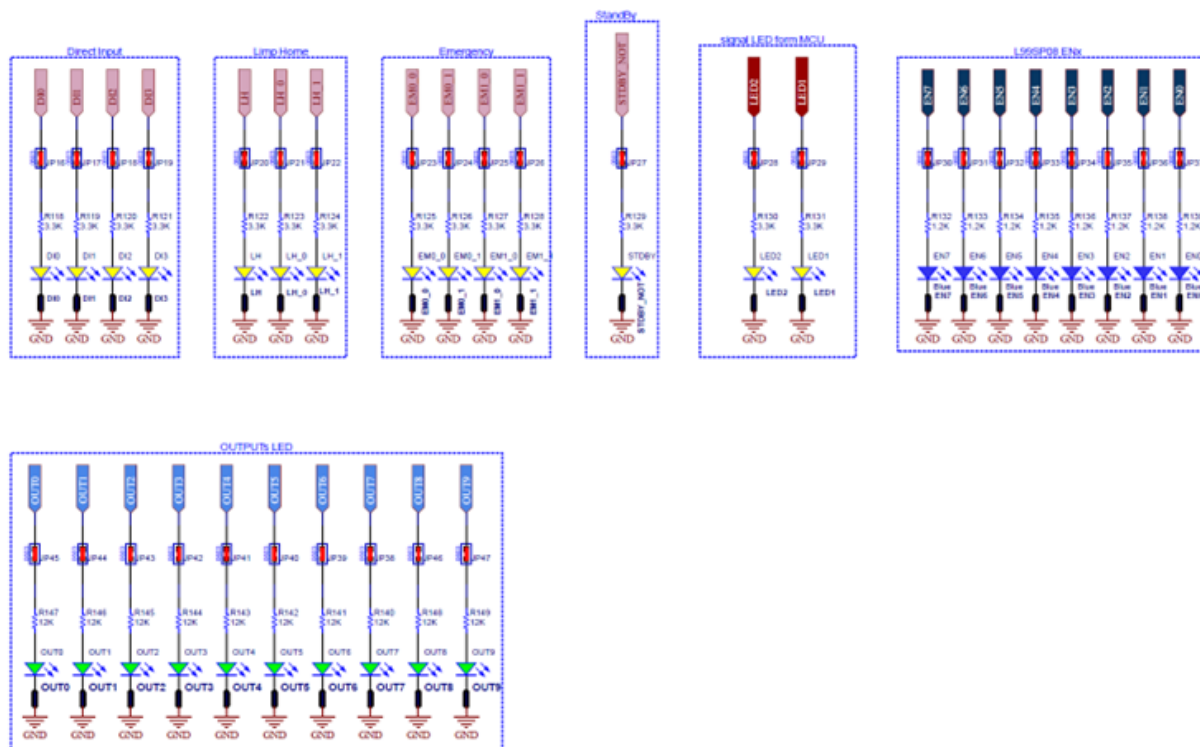
In Parking mode, an event on one of the P-channels generates a DIAG\_SP08 signal, which activates the PMIC wake-up and, subsequently, powers up the MCU.

## 2.9 Status LEDs for debugging

AEK-POW-PDUMINI is equipped with the following useful debugging and diagnostic LEDs:

- Direct input (DIx).
- Emergency (EMx).
- Limp home (LHx).
- Standby\_NOT.
- L99SP08 enable pins (ENx).
- Diagnostic (DIAGx).
- Output status LEDs (OUTx).
- MCU dedicated LEDs.

**Figure 13. AEK-POW-PDUMINI status LEDs - schematic view**



## 2.10 L99SP08 block diagram

**L99SP08** is a standby companion chip with eight smart P-channels to be coupled with STi<sup>2</sup>Fuse devices to improve standby performance. The P-channels are activated to sustain the load present in the e-fuses output by providing a lower current capability while the main e-fuses are switched off in standby condition. For each P-channel, if **L99SP08** senses a higher current request from the load, it requests the system to turn it on to perform the swap between P-channel and e-fuse.

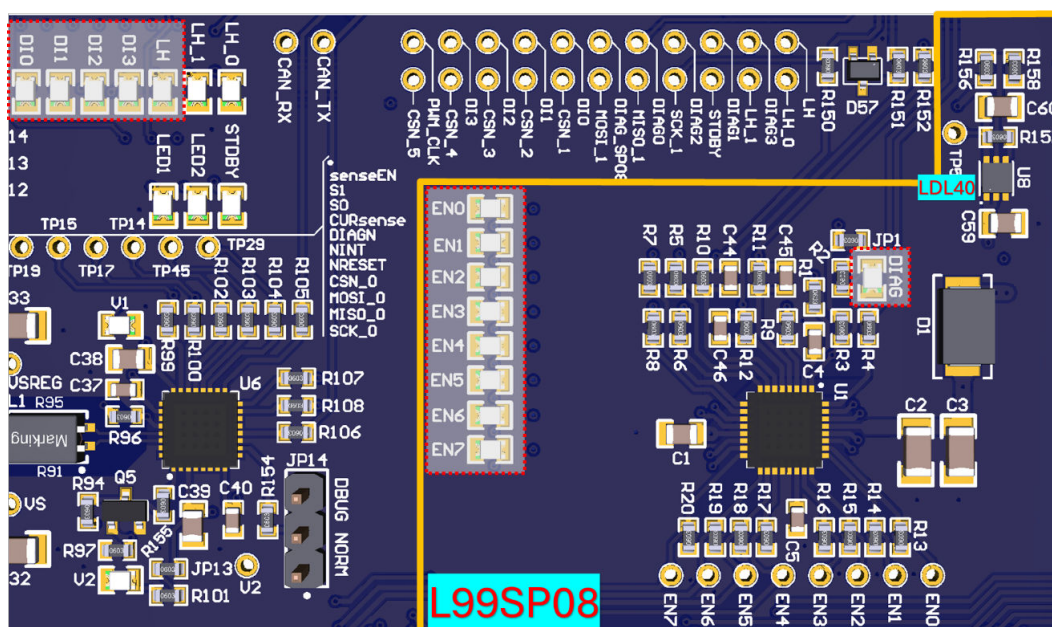
The DIAG pin present in the **L99SP08** device can be used as a wake up signal for the system because it is connected to the main PMIC (**SPSB081**). In turn, the PMIC wakes up the MCU and, when the MCU wakes up, it performs the check on the channels before switching the P-channel off and the main e-fuse on.

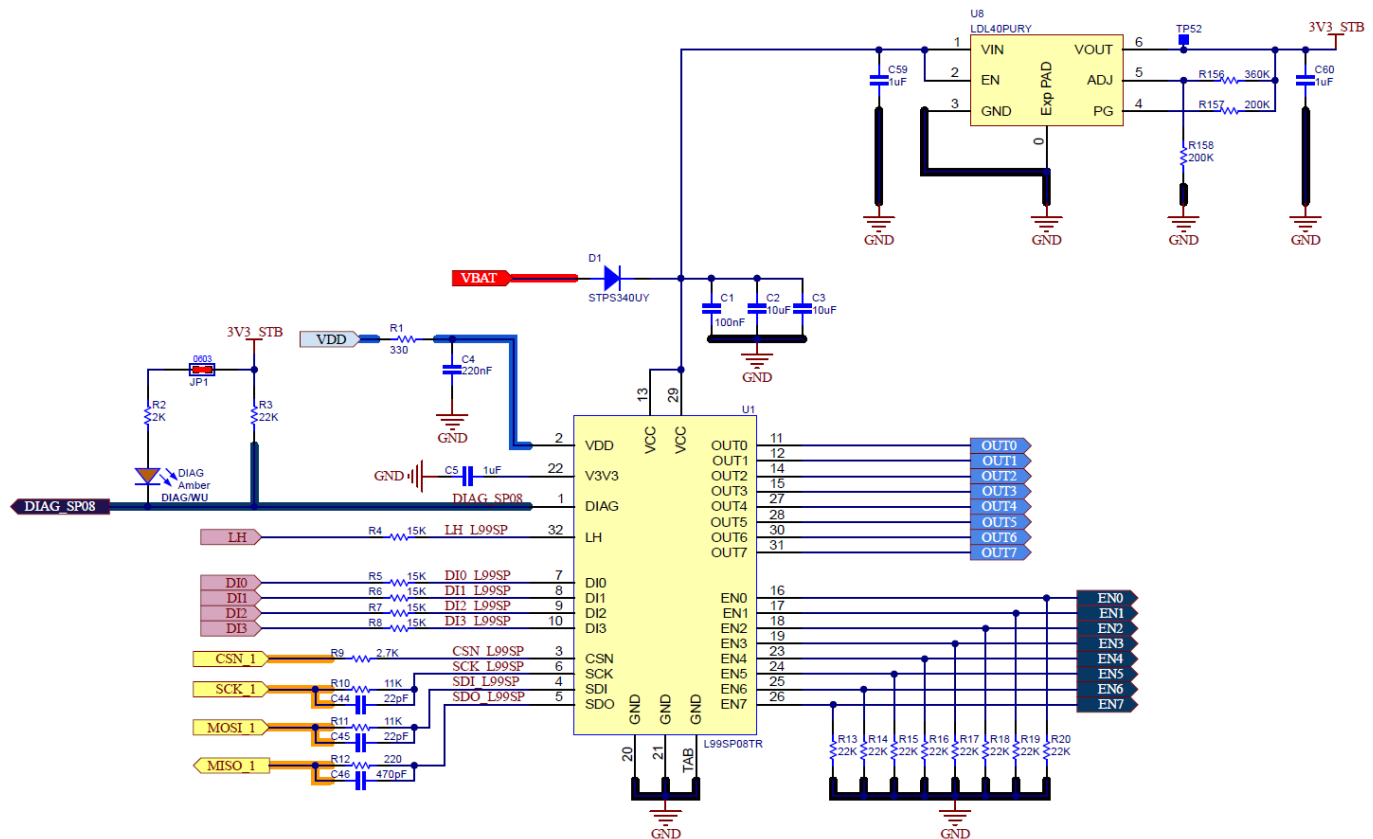
**L99SP08** features an EN pin per channel to enable external devices, like STi<sup>2</sup>Fuse. If the output current is higher than the ITHRx threshold, the enable pin (ENx) of channel x is raised, and the related P-channel is switched off.

The DIAG pin used as a wake up signal for the system is an open drain output active low that is pulled up by the **LDL40** to adapt the DIAG\_SP08 signal (3.3 V) to a level compatible with the **SPSB081** PMIC.

In fail-safe or Normal mode, outputs can be activated or deactivated directly via dedicated DLx input pins managed by the MCU.

**Figure 14. L99SP08 and LDL40 board view and status LEDs: ENx, LH, DLx, and DIAG**



**Figure 15. L99SP08 and LDL40 - schematic view**


## 2.11 STi<sup>2</sup>Fuse devices

The STi<sup>2</sup>Fuse series includes automotive smart high-side switch controllers, monolithic drivers, and system-in-package devices with controllers and power MOSFETs in the same package. These resettable electronic fuses feature a fully programmable I<sup>2</sup>t protection mechanism that ensures exceptional system flexibility and active power management. Designed to maintain vehicle power network stability and protect wires, traces, and connectors, STi<sup>2</sup>Fuse devices are essential for autonomous protection, predictive maintenance, and enhanced resilience. These features collectively contribute to a significant reduction in harness weight.

STi<sup>2</sup>Fuse are electronic fuses (e-fuses), that is, advanced electronic protection devices designed to protect circuits and components from overcurrent and overvoltage conditions. Unlike traditional fuses, which are one-time-use components that physically break to interrupt the circuit, e-fuses are integrated circuits that can electronically monitor and control the current flow, allowing reset and reuse. When an overload occurs, these devices limit the output current to a safe value defined by the user. If the overload persists, the devices enter open state, disconnecting the load from the power supply. The parameters of the I<sup>2</sup>t curve can be individually configured for each channel using two parameters: I<sub>nom</sub> (nominal current) and T<sub>nom</sub> (nominal time) with three dedicated bits for each, programmable via SPI.

Devices can operate in either latch-off or auto-retry modes following a fault. In latch-off mode, the e-fuse permanently breaks the current flow until a manual reset is performed. This means that, even if the fault is recovered, an external action (for example, a manual reset) is required. Instead, in auto-retry mode, the e-fuse temporarily interrupts the current flow and automatically attempts to restore it until the fault is recovered. e-fuses communicate the fault status to the system controller through a fault pin (DIAGx signal). The main advantages of e-fuses are:

- Resettable output.
- No need for a replacement after a fault.
- Auto-retry or latch-off modes for reset.
- Reduced cable length.
- Improved fuse time-current characteristics.
- Less variability.



The **AEK-POW-PDUMINI** hosts three types of high-side drivers based on ST proprietary **Sti<sup>2</sup>Fuse** technology:

- **VNF9D1M5Q**: It features 2 outputs, 1.5 mΩ per output, currents up to 27 A and integrated MOSFETs. It operates via SPI bus. It is protected against voltage transients on the VCC pin. Programming, control, and diagnostics are enabled by the SPI bus. Digital current sense feedback for each channel is provided through an integrated 10-bit ADC. Dedicated trimming bits allow to adjust the ADC reference current. The outputs are controllable both via SPI and via 2-OTP assignable direct inputs. Real-time diagnostic is available through the SPI bus (communication error, power limitation, overtemperature, harness protection, overload protection, VCC and output voltage monitoring, and TC monitoring). Output overcurrent detection protects the device in overload condition and is configured as latched off in thermal shutdown. The device enters a limp-home mode in case of reset digital memory, watchdog monitoring time-out event or when limp-home (LH) pin is set to high. In limp-home mode, each output is controlled by dedicated direct inputs. Emergency pins allow a rapid switch-off of the output channels in case of critical events in Normal operation mode, bypassing SPI control.
- **VNF9D5SF**: It features 2 outputs, 5.9 mΩ per output, currents up to 15 A and integrated MOSFETs. It is protected against voltage transients on the VCC pin. Programming, control, and diagnostics are enabled by the SPI bus. Digital current sense feedback for each channel is provided through an integrated 10-bit ADC. Dedicated trimming bits allow to adjust the ADC reference current. The outputs are controllable via both SPI and 2-OTP assignable direct inputs. Real-time diagnostics are available through the SPI bus (open load, output short to VCC, overtemperature, power limitation, or latch off). The device detects open-load in off-state conditions. The device enters a Fail-safe mode in case of communication loss with the microcontroller, reset of digital memory, or watchdog monitoring time-out event. In Fail-safe mode, the outputs are controlled by assignable direct inputs. The device features an operative condition called capacitive charging mode (CCM), which is available in both fail-safe and normal device states and with channels configured in bulb mode.
- **VNF9Q20SF**: It features 4 outputs, 21.5 mΩ per output, current up to 6 A, and integrated MOSFETs. It is protected against voltage transients on the VCC pin. Programming, control, and diagnostics are enabled by the SPI bus. Digital current sense feedback for each channel is provided through an integrated 10-bit ADC. Dedicated trimming bits allow to adjust the ADC reference current. The outputs are controllable via both SPI and 2-OTP assignable direct inputs. Real-time diagnostics are available through the SPI bus (open load, output short to VCC, overtemperature, power limitation, or latch off). The device detects open-load in off-state conditions. The device enters a Fail-safe mode in case of communication loss with the microcontroller, reset of digital memory, or watchdog monitoring time-out event. In Fail-safe mode, the outputs are controlled by assignable direct inputs. The device features an operative condition called capacitive charging mode (CCM), which is available in both fail-safe and normal device states and with channels configured in bulb mode.

### 2.11.1 VNF9D1M5Q block diagram

**AEK-POW-PDUMINI** includes two **VNF9D1M5Q** devices, whose outputs (OUT0, OUT1, OUT2, OUT3) are companion with the smart **L99SP08** P-channel. The SPI interface is shared among the devices and **L99SP08**, as well as the other **Sti<sup>2</sup>Fuse** devices on the board.

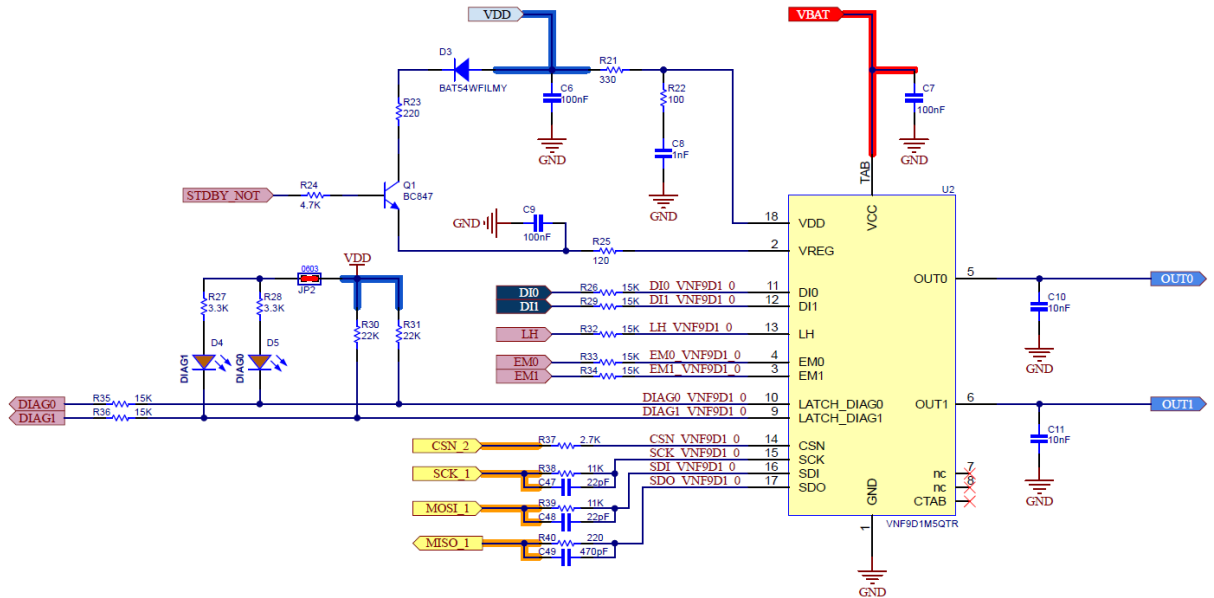
The **STDBY\_NOT** signal, which is shared with all onboard **Sti<sup>2</sup>Fuse**, is managed by the MCU. The transition from low to high determines the switch from Standby to fail-safe mode for the device. In this condition, the outputs are switched on according to the direct input (DIx) signals, controlled by the ENx of the **L99SP08**.

DIx signals control the switching on and off for two outputs in both fail-safe and Normal mode.

Emergency (EMx) pins allow a rapid switch-off of the output channels in case of critical events in Normal operation mode, bypassing SPI control. The LH signal, managed by the MCU, instructs the device to switch from Normal mode to fail-safe mode.

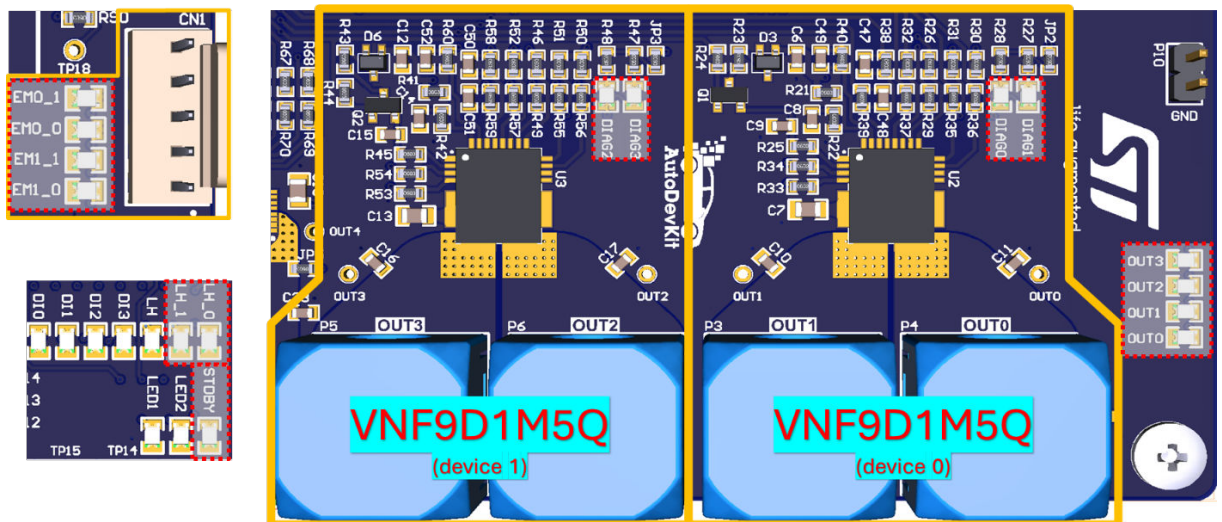
DIAG0 and DIAG1 signals (each for its respective channel) indicate a fault condition to the MCU: junction overtemperature, overcurrent protection, and hardware protection in case of I<sup>2</sup>t protection intervention.

Figure 16. VNF9D1M5Q - schematic view



AEK-POW-PDUMINI features status LEDs for debugging: STDBY\_NOT, LHx (devices 0 and 1), EMx (devices 0 and 1), DIAGx (devices 0 and 1), and OUTx (outputs).

Figure 17. VNF9D1M5Q (device 0 e 1) board view and status LEDs: OUTx, STDBY, LH, EMx, and DIAGx



### 2.11.2

#### VNF9D5SF block diagram

AEK-POW-PDUMINI includes one VNF9D5SF device, which outputs (OUT4, OUT5) are companion with the smart L99SP08 P-channel. The SPI interface is shared among the devices and L99SP08, as well as the other STi<sup>2</sup>Fuse devices on the board.

The STDBY\_NOT signal, which is shared with all STi<sup>2</sup>Fuses on board, is managed by the MCU. The transition from low to high determines the switch from Standby to fail-safe mode for the device. In this condition, the outputs are switched on according to the DIx signals, controlled by the ENx of the L99SP08. Direct input signals control the switching on and off for two outputs in both fail-safe and Normal mode.

VNF9D5SF has a PWM\_CLK input pin managed by the MCU timer to control the outputs in PWM mode. In this mode, the outputs switch at a frequency equal to the ratio between PWM\_CLK and the combination of 2 bits (whose values can be 512, 1024, 2048, 4096), that is, setting a ratio of 1024 results at an output frequency of 390 Hz (400 kHz/1024). The duty cycle of the outputs can also be set with a granularity of 10 bits, ranging from 0% (000) to 100% (3FF).

Figure 18. VNF9D5SF - schematic view

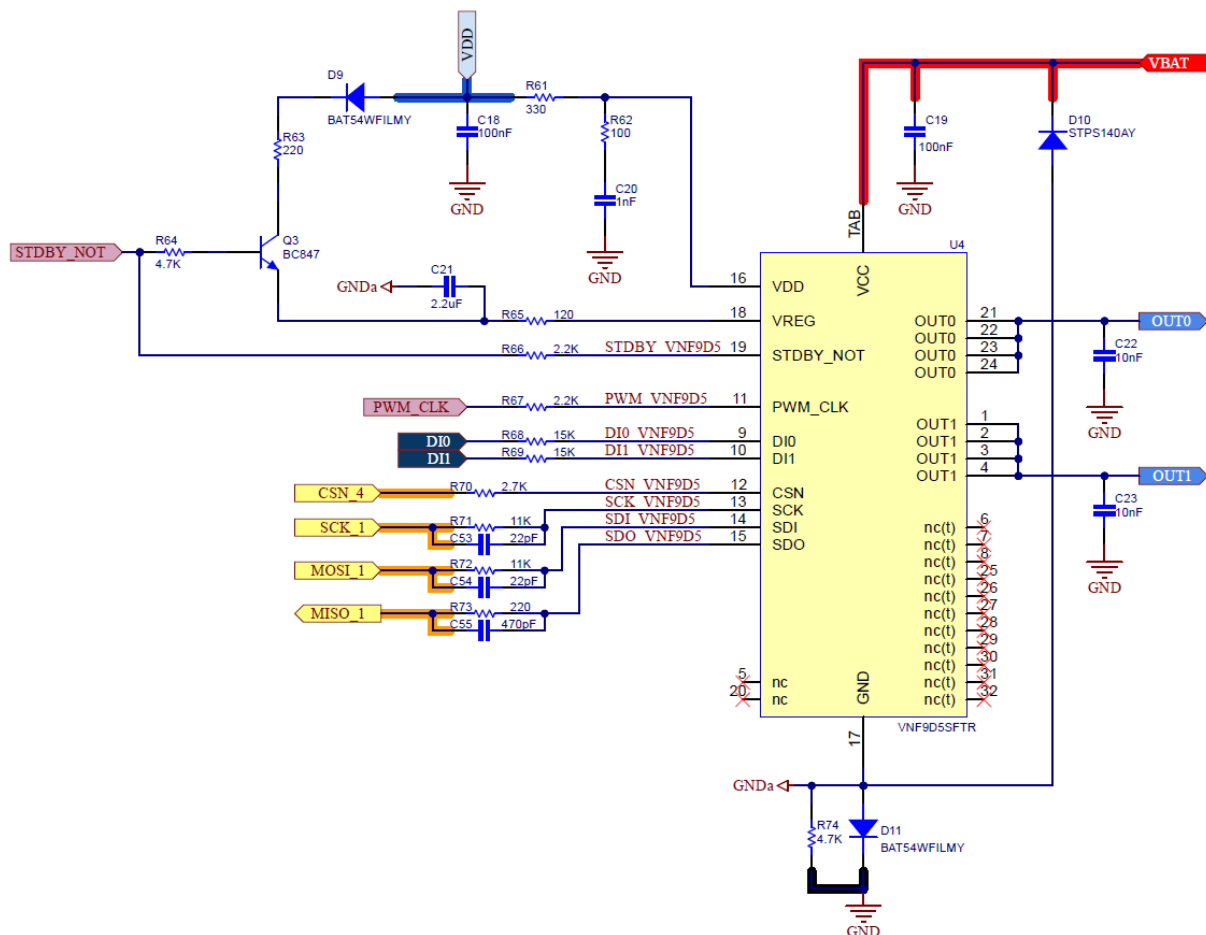
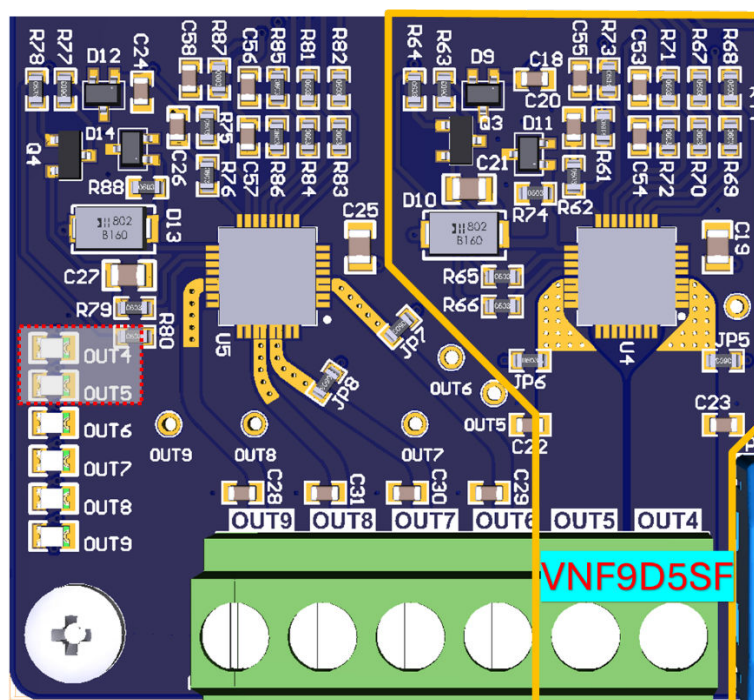


Figure 19. VNF9D5SF board view and output status LEDs: OUT4 and OUT5

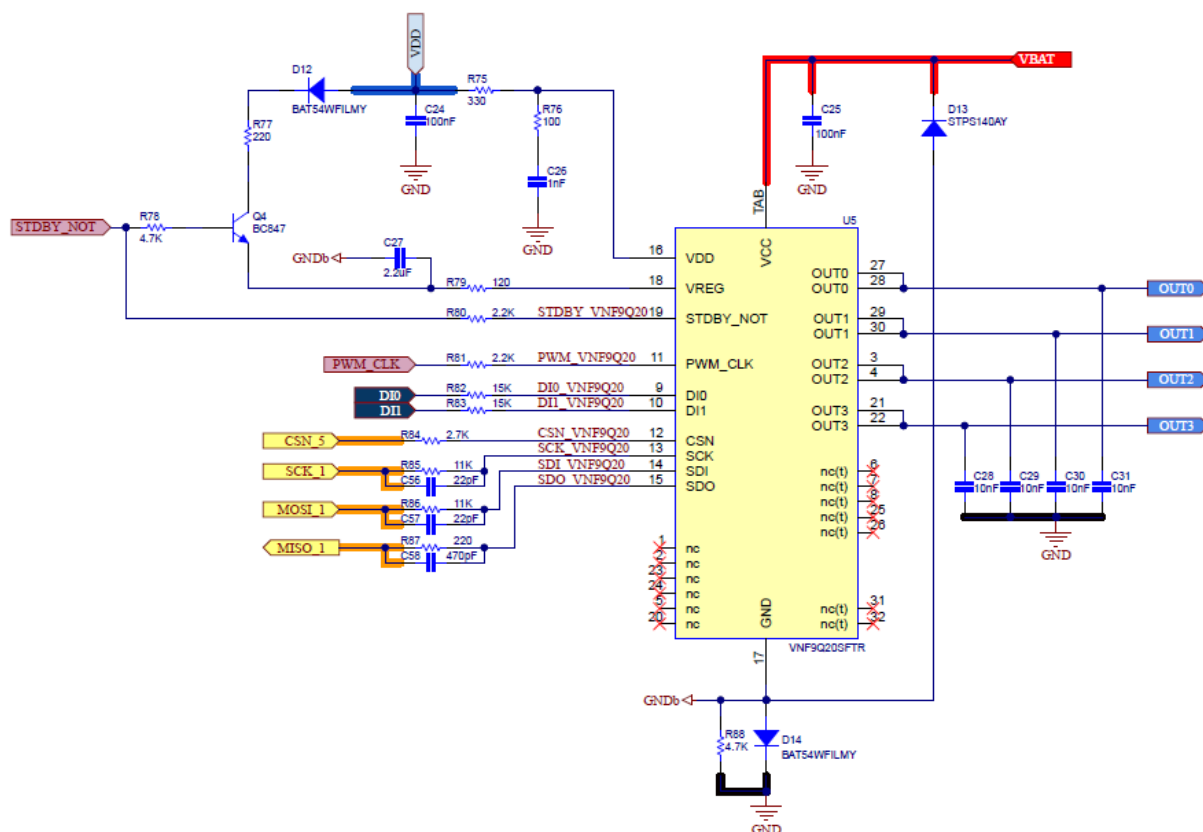


### 2.11.3 VNF9Q20SF block diagram

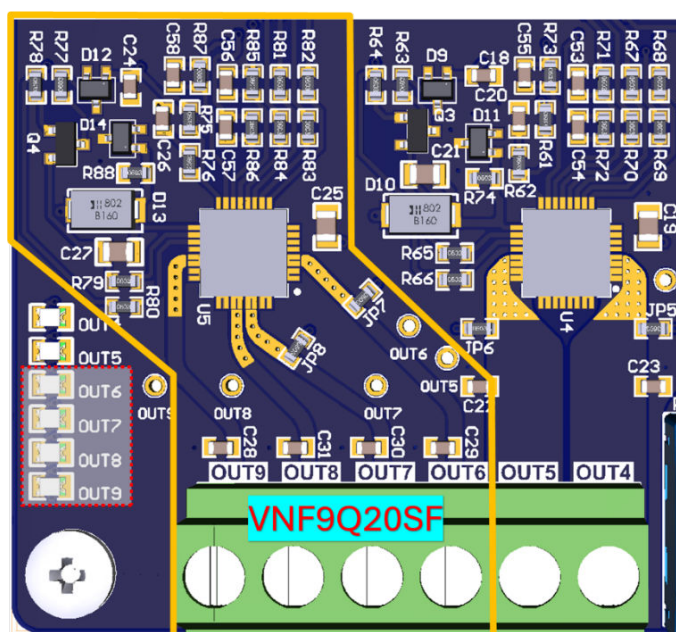
AEK-POW-PDUMINI includes a VNF9Q20SF with four outputs: outputs OUT6 and OUT7 are companion with the smart L99SP08 P-channel, while OUT8 and OUT9 are managed by the MCU via SPI.

This device has the same features of VNF9D5SF, as described above in Section 2.11.2.

**Figure 20. VNF9Q20SF - schematic view**



**Figure 21. VNF9Q20SF board view and output status LEDs: OUT6, OUT7, OUT8, OUT9**





## 2.12 SPSB081 block diagram

**SPSB081** is a power management system IC providing power management functionality, including various Standby modes to minimize the power consumption with programmable local and remote wake-up capability, as well as LIN and CAN-FD physical communication layers.

The device has one low-drop voltage regulator to supply the system microcontroller and one voltage tracker to supply external peripheral loads such as sensors. V1 is available with a fixed rail (5 V or 3.3 V) and V1 overvoltage detection and protection solution, while V2 is a tracker voltage regulator of V1, programmable by SPI with 5 V or 3.3 V. Moreover, the device features four high-side drivers to supply LEDs and sensors. All outputs are short-short-circuit (SC) protected and implement open-load diagnosis. It uses a dedicated SPI interface that allows device control and diagnostics.

The master role of this device is to handle wake up from several sources:

- Internal wake up capability based on CAN bus activity due to the CAN embedded physical layer.
- External wake up capability coming from the [L99SP08](#) companion IC when the current request threshold is overcome. The DIAG signal from [L99SP08](#) P-channel is used as wake up for the [SPSB081](#) (WU1 input).

**Figure 22. SPSB081 - board view**

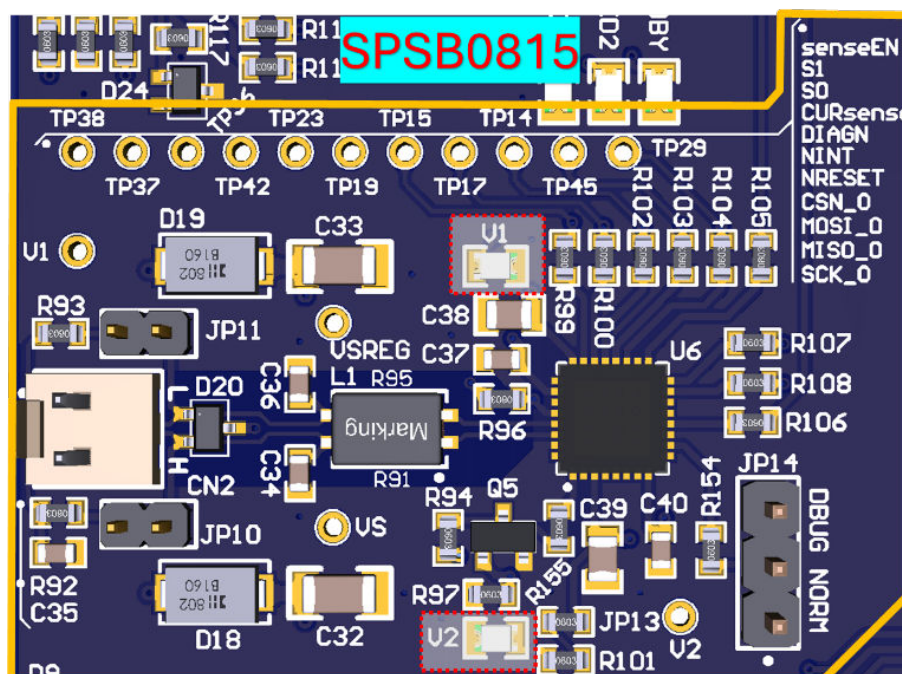
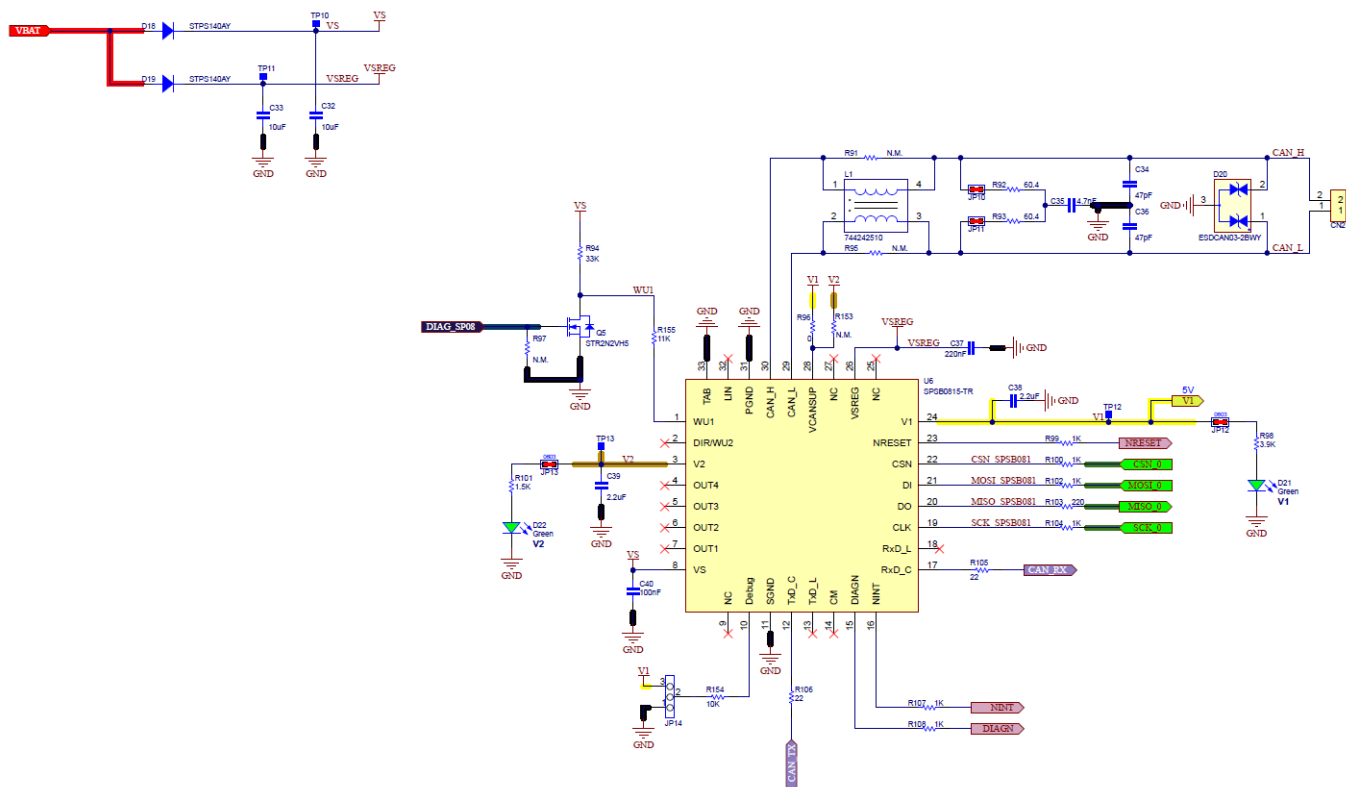


Figure 23. SPSB081 - schematic view



## 2.13 VN7140AJ block diagram

VN7140AJ is a high-side driver with MultiSense analog feedback for automotive applications, with a typical  $R_{ds(on)}$  of 120 m $\Omega$ .

The master role is to power the MCU board with 12 V (VBAT) upon powering up or following a wake up event. In fact, the V1 voltage (SPSB081) is used to enable the VN7140AJ output (via the INPUT pin).

Figure 24. VN7140AJ - board view

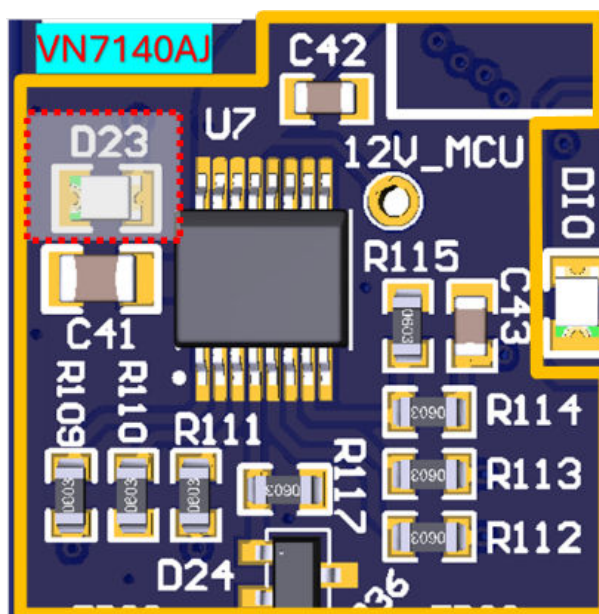
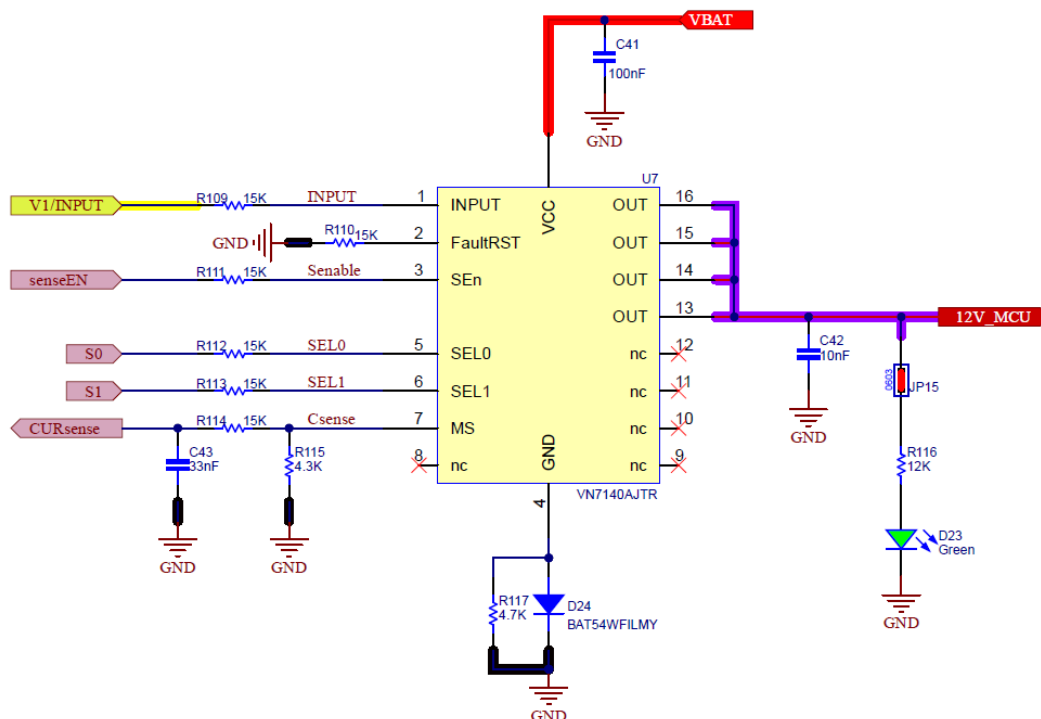


Figure 25. VN7140AJ - schematic view



## 3 Hardware test results

The purpose of this evaluation was to validate the functionality of all hardware components of **AEK-POW-PDUMINI**.

### 3.1 Test setup

This section lists the test equipment used. The test setup for individual test might slightly differ from the schematic (different output used, etc.).

### 3.2 Used tools

**Table 2. Used tools**

| Tool            | Description                                               |
|-----------------|-----------------------------------------------------------|
| Electronic load | Chroma 63640-80-80 400 W<br>Chroma 63630-80-60 300 W      |
| Power supply    | EA PSI 10080-120 2U 3 KW                                  |
| Oscilloscope    | R&S MXO5                                                  |
| Current probes  | R&S RT-ZC20B                                              |
| Board           | AEK-POW-PDUMINI                                           |
| MCU Board       | AEK-MCU-C4MLIT2 plugged on unified connector with 96 pins |

Loads: electronic loads.

Default conditions:

- VCC: 14 V.
- Temperature: ambient =  $23 \pm 5$  °C.

### 3.3 Sleep, wake-up, startup time conditions

#### 3.3.1 Current consumption

This test was performed on a modified board on which some of the indication LEDs were removed to reduce current consumption in Parking mode as much as possible.

##### Results

- Parking mode (no active outputs): below 1 mA (the hardware does not allow accurate measurements below 1 mA).
- Normal mode (no active outputs): 118 mA (**AEK-POW-PDUMINI** and **AEK-MCU-C4MLIT2** board).

#### 3.3.2 Wake-up sources and startup times

The purpose of these tests is to verify that all wake-up functions work correctly and can wake the module up. The startup times were also measured during these tests.

Two different startup times were measured:

- Time from wake-up event by reactivation from Parking mode, triggered as soon as the power demand exceeds the programmed threshold of the P-channel.
- Time from wake-up event by CAN message.

##### 3.3.2.1 Test conditions

1. Power supply: PSI10080-120 (14 V @ 80 A).
2. Electronic load: 63640-80-80 in parallel with 63630-80-60 (CCH mode).
3. MXO5 oscilloscope: voltage passive and current probes (RT-ZC20B).



### 3.3.2.2

#### OUT0

OUT0 was activated through L99SP08 and the AEK-POW-PDUMINI board was switched to Parking mode.

Then, the output was loaded with a current exceeding the set threshold (100 mA) in L99SP08, which pulled the DIAG\_SP08 signal low and activated the main power switch.

We measured the time elapsed from the moment DIAG\_SP08 signal was pulled low until CSN0 (SPI0 used for PMIC device) and CSN1 (SPI1 used for P-channel and ST<sup>2</sup>Fuse) reached a stable voltage of 5 V.

**Figure 26. OUT0 wake-up, DIAG\_SP08 signal low and CSN0 and CSN1 reach 5 V ( $t_{CSN0-1(5\text{ V})} = 211.3\text{ ms}$ )**



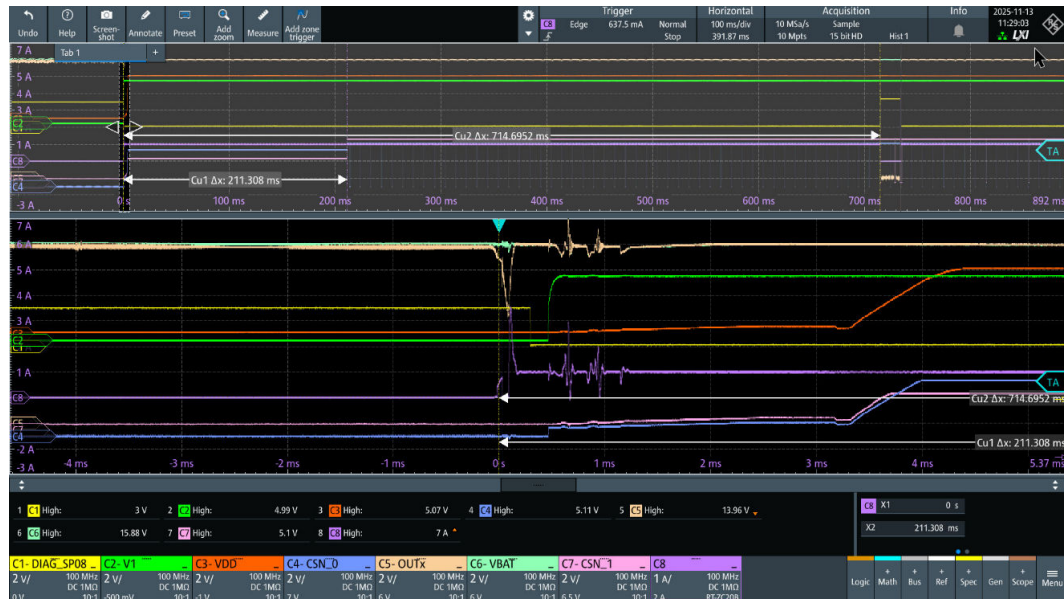
As described in Figure 26, the 5 V voltage reached by CSN0 and CSN1 is shown, as well as the start of communication on SPI0 (used for PMIC) at around 213.5 ms.

**Figure 27. OUT0 wake-up, CSN0 and CSN1 reach 5 V and CSN0 start communication**



The wake-up event is triggered by the electronic load activation configured in Current constant high (CCH) mode at 1 A. In this condition, the DIAG\_SP08 signal drops within 300  $\mu\text{s}$ , followed by the V1 signal (coming from the PMIC and used to enable VN7140AJ, which, in turn, powers the MCU board), which turns on after 500  $\mu\text{s}$ .

Figure 28. OUT0 wake-up signals detail ( $t_{DIAG\_SP08(low)} = 300 \mu s$   $t_{V1(high)} = 500 \mu s$ )



### 3.3.2.3

#### OUT2

The same test was performed on OUT2 (managed by P-channel companion with VNF9D1M5Q), obtaining the same results as in the OUT0 test.

Figure 29. OUT2 wake-up - DIAG\_SP08, CSN0, CSN1, and output signals



### 3.3.2.4

#### OUT4-5-6

The same test was performed on OUT4-5 (managed by P-channel companion with VNF9D5SF). The output was loaded with a current exceeding the set threshold (100 mA) in L99SP08, which then pulled the DIAG\_SP08 signal low and activated the main power switch.

In this test, we measured the time elapsed from the moment DIAG\_SP08 signal was pulled low until CSN0 (SPI0 used for PMIC device) and CSN1 (SPI1 used for P-channel and STi<sup>2</sup>Fuse) reached a stable voltage of 5 V.

Figure 30. OUT4 wake-up (tCSN0-1(5 V) = 210.43 ms)



The same result was obtained on OUT5 by performing the same test.

Figure 31. OUT5 wake-up



### 3.3.2.5

#### CAN wake-up

In this test, the module was set to Parking mode with all outputs disabled. The module was then woken up with a CAN message.

This test measured the time elapsed from the start of the CAN message until CSN0 (SPI0 used for PMIC device) and CSN1 (SPI1 used for P-channel and ST<sup>2</sup>Fuse) reached a stable voltage of 5 V.

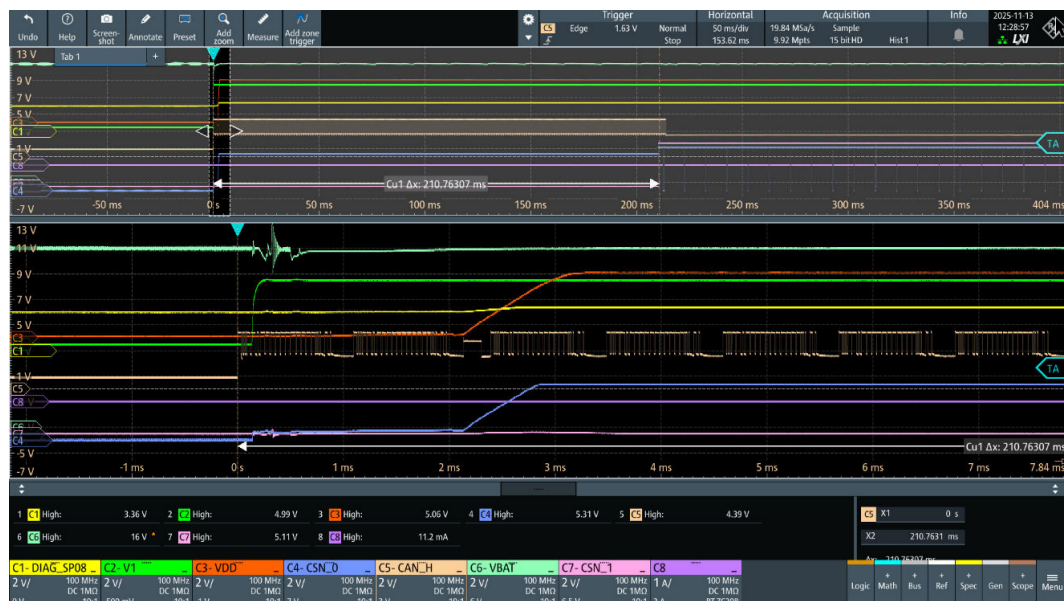


Figure 32. Module wake-up through CAN ( $t_{CSN0-1(5V)} = 210.73 \text{ ms}$ )

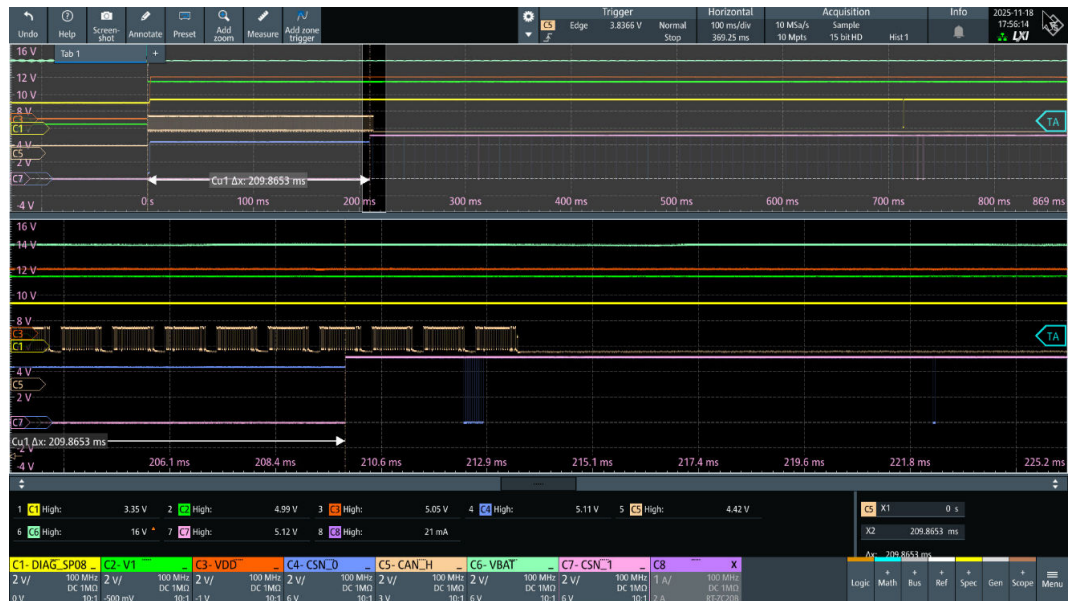


The wake-up event is triggered by sending a CAN message. In this condition, the V1 signal (coming from the PMIC and used to enable VN7140AJ, which powers the MCU board) turns on after 100  $\mu\text{s}$ .

Figure 33. CAN wake-up signals detail



As previously described, the 5 V voltage reached by CSN0 and CSN1 is shown, as well as the start of communication on SPI0 (used for PMIC) at around 209.86 ms.

**Figure 34. CAN wake-up, CSN0 and CSN1 reach 5 V and CSN0 startup communication**


### 3.4 Regular I<sup>2</sup>t functionality

These tests verified the functionality of the wiring protection (STi<sup>2</sup>Fuse) when switching from Parking mode to Normal mode on the [AEK-POW-PDUMINI](#) board. After switching the board on, a CAN message was sent to activate each tested output. Then, the board was set to Parking mode.

**Notice:** Each test was performed at least three times.

#### 3.4.1 Test conditions

1. Power supply: PSI10080-120 (14 V @ 80 A).
2. Electronic load: 63640-80-80 in parallel with 63630-80-60 (CCH mode).
3. MXO5 oscilloscope: voltage passive and current probes (RT-ZC20B).



### 3.4.2 VNF9D1M5Q

OUT0 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 21\text{ A}$  and  $t_{nom} = 1\text{ s}$ . A load current of 30 A was applied to the output using an electronic load.

Results:  $t_{latch-off} = 33.74\text{ ms}$

**Figure 35. OUT0, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ ,  $t_{nom} = 1\text{ s}$ ) (1 of 3)**



**Figure 36. OUT0, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ ,  $t_{nom} = 1\text{ s}$ ) (2 of 3)**



Figure 37. OUT0, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ ,  $t_{nom} = 1\text{ s}$ ) (3 of 3)



OUT1 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 16\text{ A}$  and  $t_{nom} = 172\text{ s}$ . A load current of 30 A was applied to the output using an electronic load.

Results:  $t_{latch-off} = 2.82\text{ s}$

Figure 38. OUT1, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ ,  $t_{nom} = 172\text{ s}$ ) (1 of 3)



Figure 39. OUT1, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ ,  $t_{nom} = 172\text{ s}$ ) (2 of 3)



Figure 40. OUT1, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ ,  $t_{nom} = 172\text{ s}$ ) (3 of 3)



OUT2 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 21\text{ A}$  and  $t_{nom} = 1\text{ s}$ . A load current of 30 A was applied to the output using an electronic load.

Results:  $t_{\text{latch-off}} = 31.65 \text{ ms}$

Figure 41. OUT2, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{\text{nom}} = 21 \text{ A}$ ,  $t_{\text{nom}} = 1 \text{ s}$ ) (1 of 3)

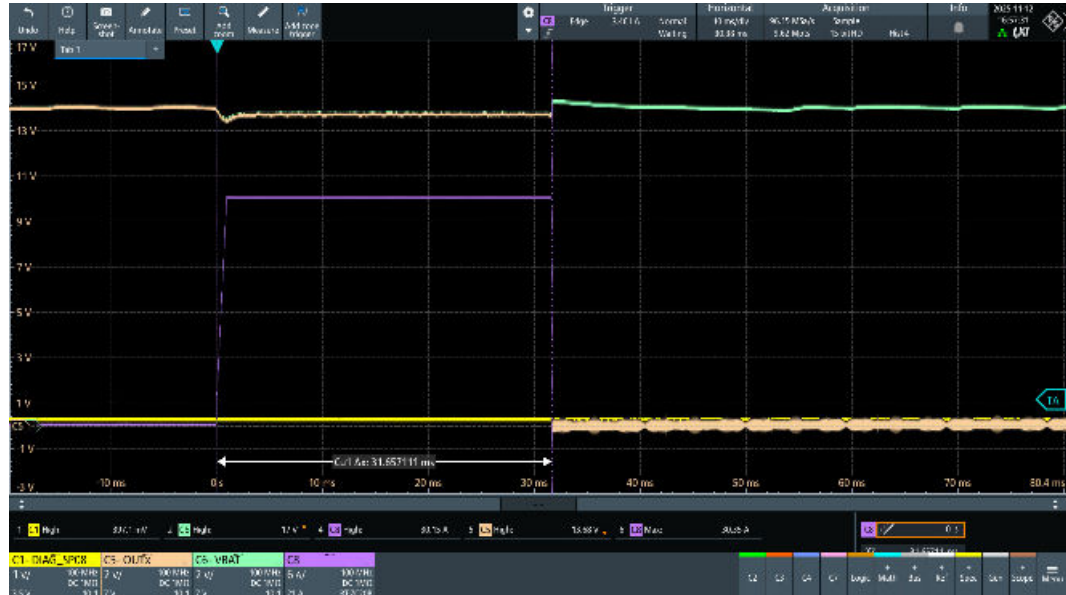


Figure 42. OUT2, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{\text{nom}} = 21 \text{ A}$ ,  $t_{\text{nom}} = 1 \text{ s}$ ) (2 of 3)





Figure 43. OUT2, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ ,  $t_{nom} = 1\text{ s}$ ) (3 of 3)



OUT3 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 16\text{ A}$  and  $t_{nom} = 172\text{ s}$ . A load current of 30 A was applied to the output using an electronic load.

Results:  $t_{latch-off} = 2.63\text{ s}$

Figure 44. OUT3, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ ,  $t_{nom} = 172\text{ s}$ ) (1 of 3)

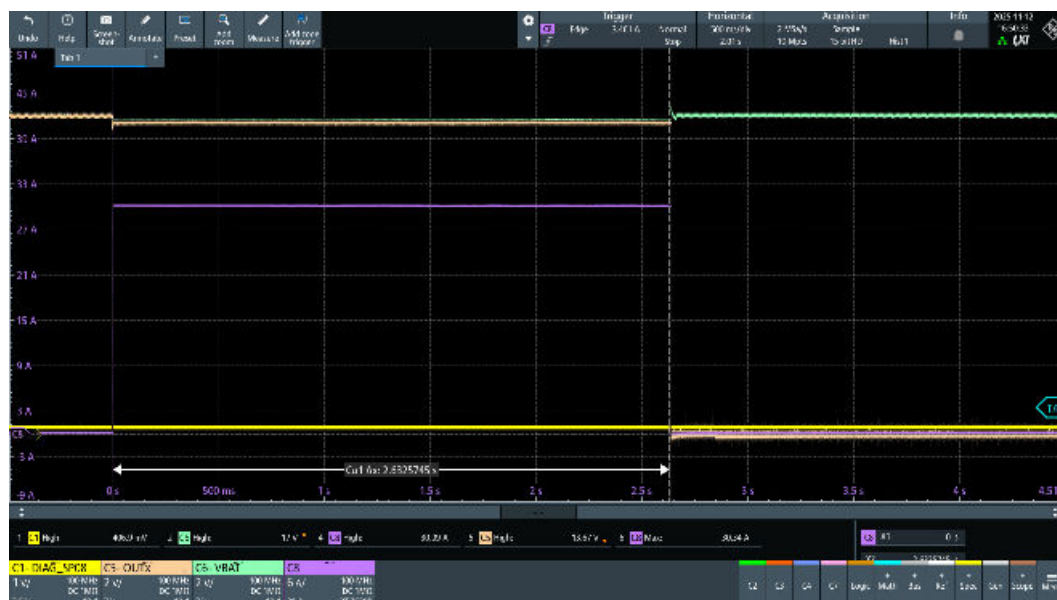


Figure 45. OUT3, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ ,  $t_{nom} = 172\text{ s}$ ) (2 of 3)

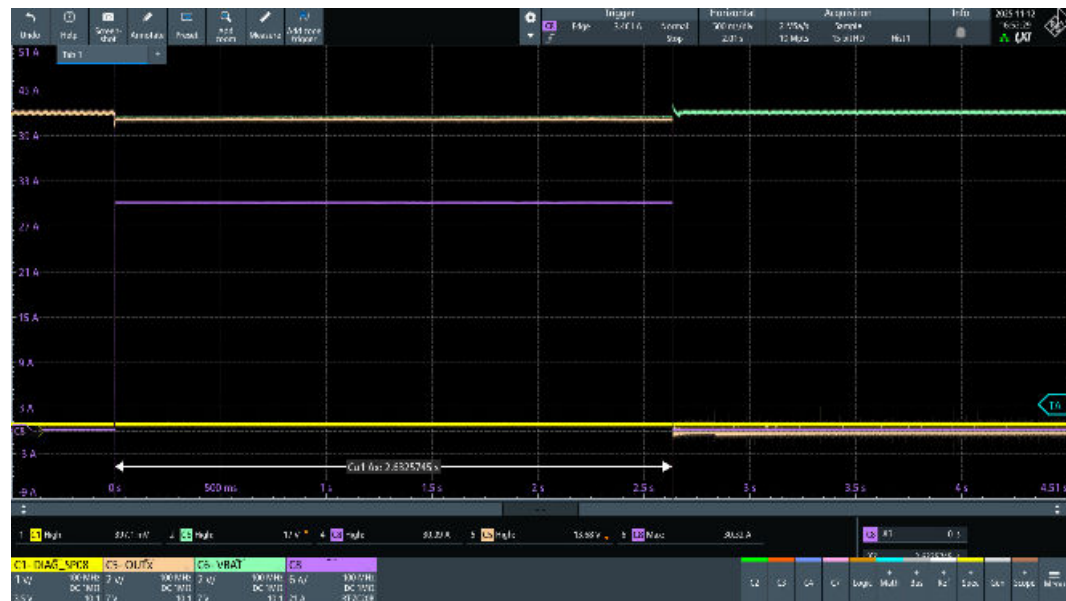


Figure 46. OUT3, VNF9D1M5Q STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ ,  $t_{nom} = 172\text{ s}$ ) (3 of 3)



### 3.4.3 VNF9D5SF

OUT4 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 4\text{ A}$  and  $t_{nom} = 44\text{ s}$ . A load current of 10 A was applied to the output using an electronic load.

Results:  $t_{latch-off} = 344.86\text{ ms}$

**Figure 47. OUT4, VNF9D5SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 4\text{ A}$ ,  $t_{nom} = 44\text{ s}$ ) (1 of 3)**



**Figure 48. OUT4, VNF9D5SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 4\text{ A}$ ,  $t_{nom} = 44\text{ s}$ ) (2 of 3)**



The screenshot displays a Keysight Oscilloscope interface. The main window shows a current waveform (labeled 'I1') with a high-frequency tail. The y-axis represents current in Amperes (A), ranging from -5 to 17. The x-axis represents time in milliseconds (ms), ranging from 0 to 854. A measurement bar indicates a duration of 344.657 ms. The top bar shows various settings like Input, Edge, Delay, and Sample Rate.

**Results:  $t_{\text{latch-off}} = 3.35 \text{ s}$**



Figure 51. OUT5, VNF9D5SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 10\text{ A}$ ,  $t_{nom} = 214\text{ s}$ ) (2 of 3)

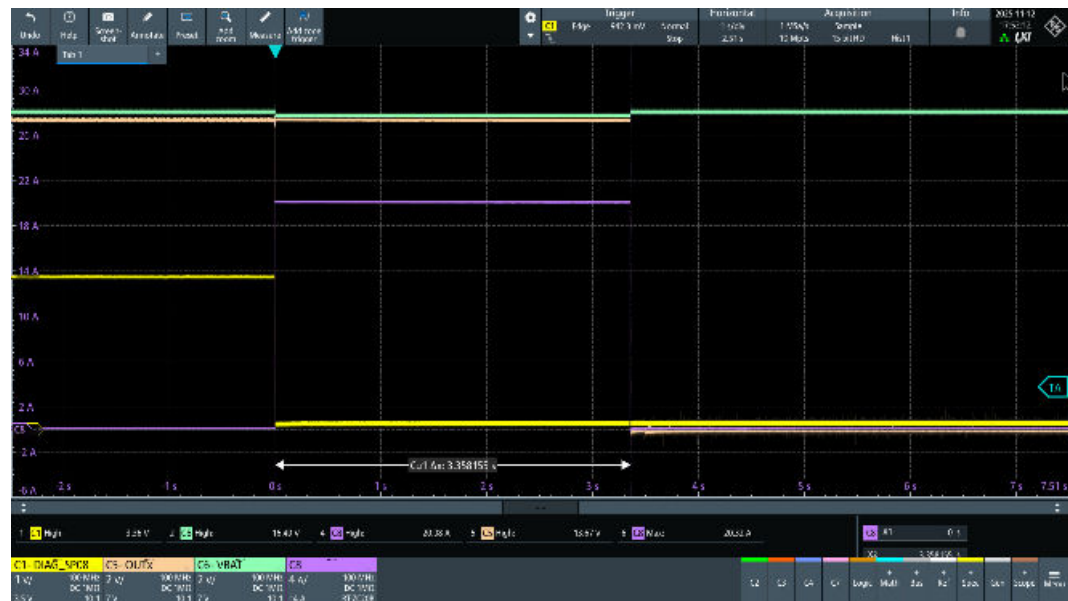


Figure 52. OUT5, VNF9D5SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 10\text{ A}$ ,  $t_{nom} = 214\text{ s}$ ) (3 of 3)

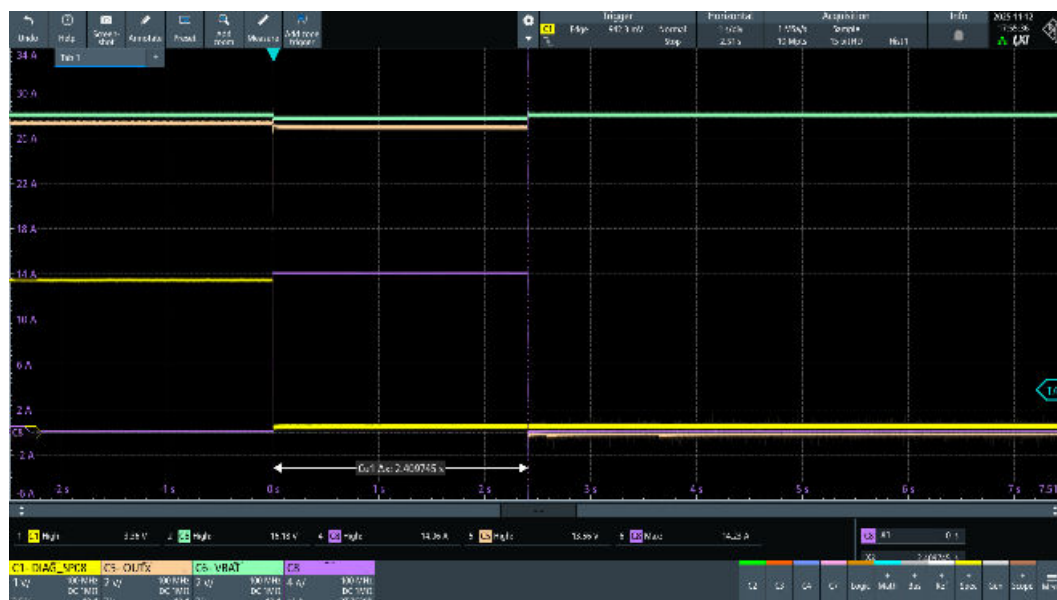


### 3.4.4 VNF9Q20SF

OUT6 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 6\text{ A}$  and  $t_{nom} = 300\text{ s}$ . A load current of 14 A was applied to the output using an electronic load.

Results:  $t_{latch-off} = 2.4\text{ s}$

**Figure 53. OUT6, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (1 of 3)**



**Figure 54. OUT6, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (2 of 3)**



Figure 55. OUT6, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (3 of 3)



OUT7 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 6\text{ A}$  and  $t_{nom} = 300\text{ s}$ . A load current of 14 A was applied to the output using an electronic load.

Results:  $t_{latch-off} = 2.4\text{ s}$

Figure 56. OUT7, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (1 of 3)

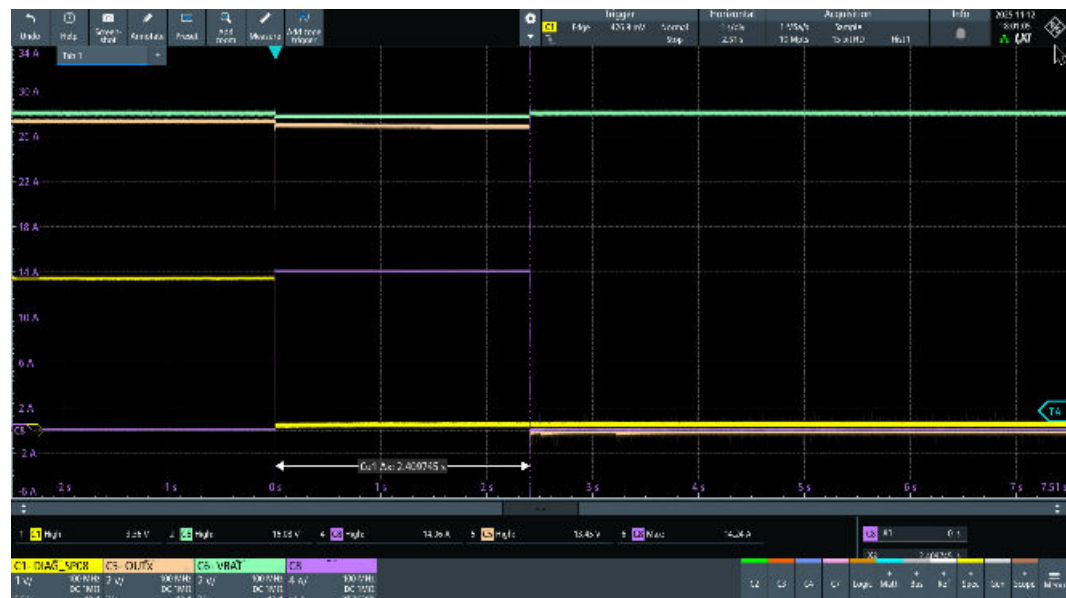


Figure 57. OUT7, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (2 of 3)

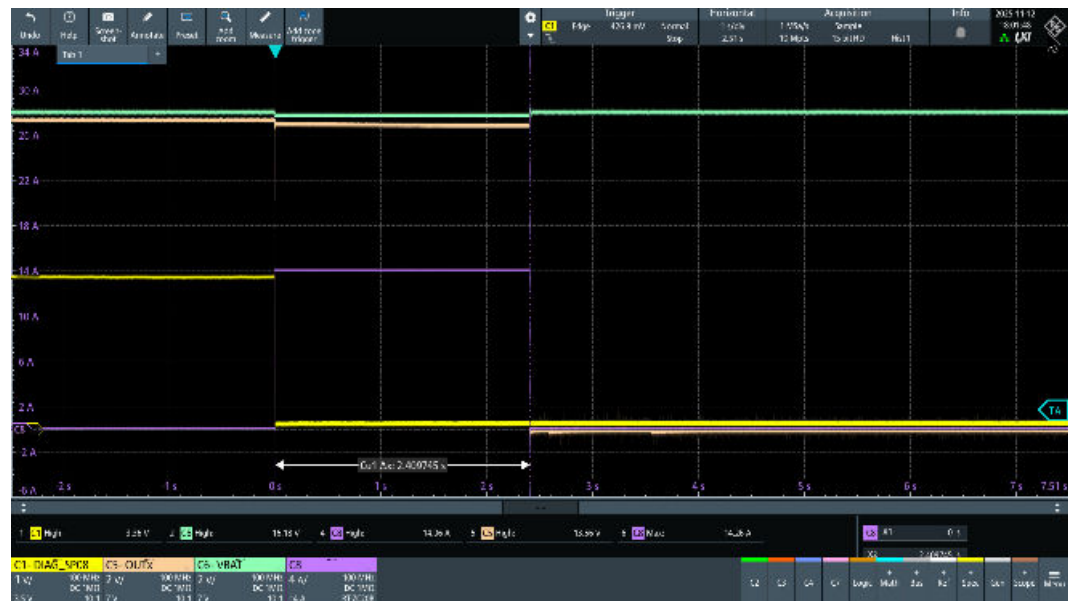
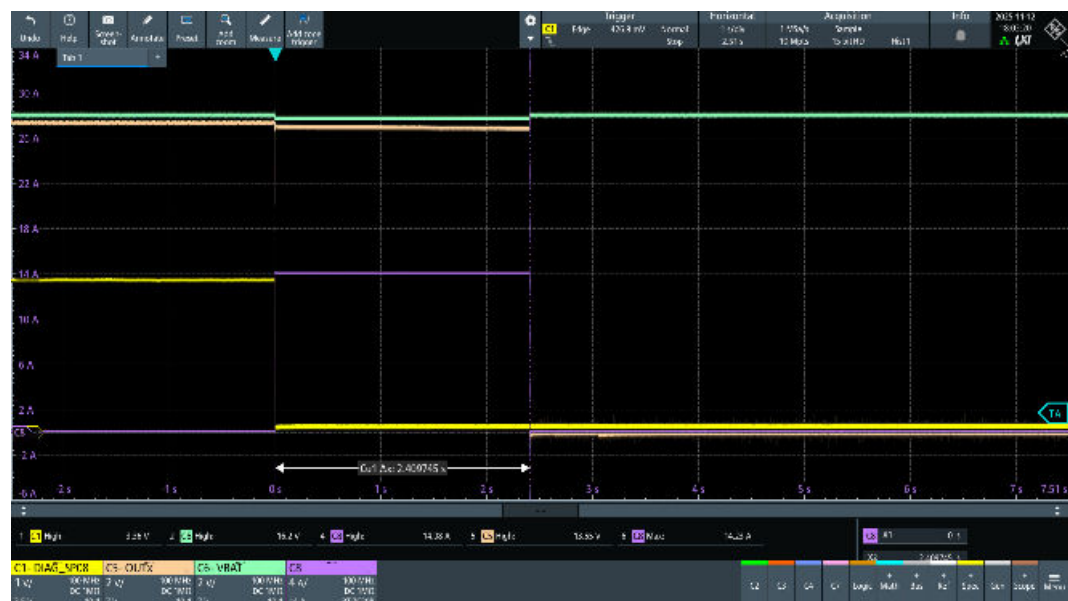


Figure 58. OUT7, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (3 of 3)



OUT8 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 6\text{ A}$  and  $t_{nom} = 300\text{ s}$ . A load current of 14 A was applied to the output using an electronic load.

This output is not companioned with L99SP08, it is managed by the MCU via SPI.



Results:  $t_{\text{latch-off}} = 2.4$  s

Figure 59. OUT8, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{\text{nom}} = 6$  A,  $t_{\text{nom}} = 300$  s) (1 of 3)



Figure 60. OUT8, VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{\text{nom}} = 6$  A,  $t_{\text{nom}} = 300$  s) (2 of 3)



Figure 61. OUT8, VNF9Q20SF STI<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (3 of 3)



OUT9 was selected for this measurement, and the programmed  $I^2t$  curve was:  $I_{nom} = 6\text{ A}$  and  $t_{nom} = 300\text{ s}$ . A load current of 15 A was applied to the output using an electronic load.

This output is not companioned with L99SP08, it is managed by the MCU via SPI.

Results:  $t_{latch-off} = 2.39\text{ s}$

Figure 62. OUT9 (@15 A load current), VNF9Q20SF STI<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (1 of 3)



Figure 63. OUT9 (@15 A load current), VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (2 of 3)



Figure 64. OUT9 (@15 A load current), VNF9Q20SF STi<sup>2</sup>Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ ,  $t_{nom} = 300\text{ s}$ ) (3 of 3)



### 3.4.5 Results' summary

**Table 3. Results' summary**

|          | VNF9D1M5Q (dev.0)                          | VNF9D1M5Q (dev.1)                            | VNF9D5SF                                   |                                              |                                            | VNF9Q20SF                                    |                                             |                                             |                                             |                                             |
|----------|--------------------------------------------|----------------------------------------------|--------------------------------------------|----------------------------------------------|--------------------------------------------|----------------------------------------------|---------------------------------------------|---------------------------------------------|---------------------------------------------|---------------------------------------------|
|          | OUT0<br>$I_{nom}=21$<br>A $t_{nom}=1$<br>s | OUT1<br>$I_{nom}=16$<br>A $t_{nom}=172$<br>s | OUT2<br>$I_{nom}=21$<br>A $t_{nom}=1$<br>s | OUT3<br>$I_{nom}=16$<br>A $t_{nom}=172$<br>s | OUT4<br>$I_{nom}=4$ A<br>$t_{nom}=44$<br>s | OUT5<br>$I_{nom}=10$<br>A $t_{nom}=214$<br>s | OUT6<br>$I_{nom}=6$ A<br>$t_{nom}=300$<br>s | OUT7<br>$I_{nom}=6$ A<br>$t_{nom}=300$<br>s | OUT8<br>$I_{nom}=6$ A<br>$t_{nom}=300$<br>s | OUT9<br>$I_{nom}=6$ A<br>$t_{nom}=300$<br>s |
|          | @ 30 A                                     |                                              | @ 30 A                                     |                                              | @ 10 A                                     | @ 20 A                                       | @ 14 A                                      | @ 14 A                                      | @ 14 A                                      | @ 15 A                                      |
| Meas. 1  | 33.74 ms                                   | 2.82 s                                       | 31.65 ms                                   | 2.63 s                                       | 344.86 ms                                  | 3.35 s                                       | 2.4 s                                       | 2.4 s                                       | 2.4 s                                       | 2.39 s                                      |
| Meas. 2  | 33.74 ms                                   | 2.82 s                                       | 31.65 ms                                   | 2.63 s                                       | 344.86 ms                                  | 3.35 s                                       | 2.4 s                                       | 2.4 s                                       | 2.4 s                                       | 2.39 s                                      |
| Meas. 3  | 33.74 ms                                   | 2.82 s                                       | 31.65 ms                                   | 2.63 s                                       | 344.86 ms                                  | 3.35 s                                       | 2.4 s                                       | 2.4 s                                       | 2.4 s                                       | 2.39 s                                      |
| Expected | 31.25 ms                                   | 2.68 s                                       | 31.25 ms                                   | 2.68 s                                       | 343.75 ms                                  | 3.34 s                                       | 2.34 s                                      | 2.34 s                                      | 2.34 s                                      | 2.34 s                                      |

All measured latch-off times are within the expected ranges.

## 3.5 Maximum current on all channels

This test was meant to evaluate the thermal design of the **AEK-POW-PDUMINI** board under high current load. Test results were recorded with a thermal imaging camera.

### 3.5.1 Test conditions

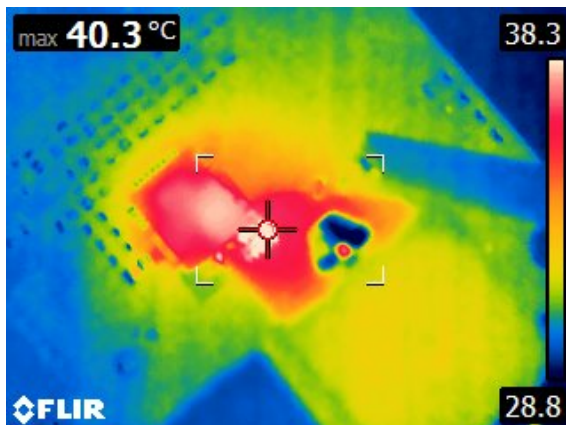
1. Power supply: PSI10080-120 (14 V @ 80 A).
2. Electronic load: 63640-80-80 and 63630-80-60 in single mode (both CCH mode).
3. MXO5 oscilloscope: voltage passive and current probes (RT-ZC20B).
4. Thermal imaging FLIR ETS320.

### 3.5.2 Individual channels

In this test, channels were loaded to their maximum rated current, one at a time. The output was connected to the electronic load for about 300 secs. For each output, at least three temperatures were acquired at different times: at 5 secs (startup), after 160 secs and after 300 secs (measurement stopped).

#### VNF9D1M5Q at 26 A (OUT0)

**Figure 65. VNF9D1M5Q @ 26 A (OUT0) - Thermal image: 5 secs**



**Figure 66. VNF9D1M5Q @ 26 A (OUT0) - Thermal image: 160 secs**

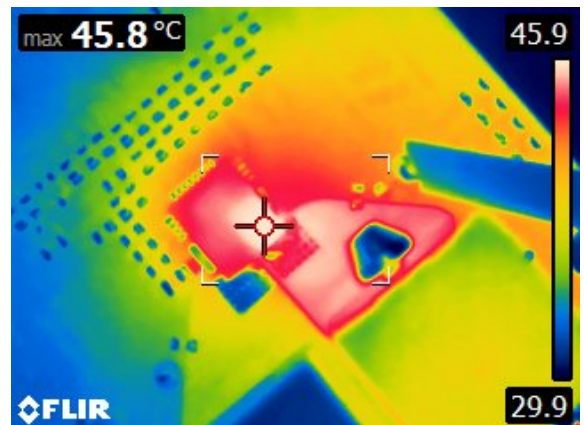
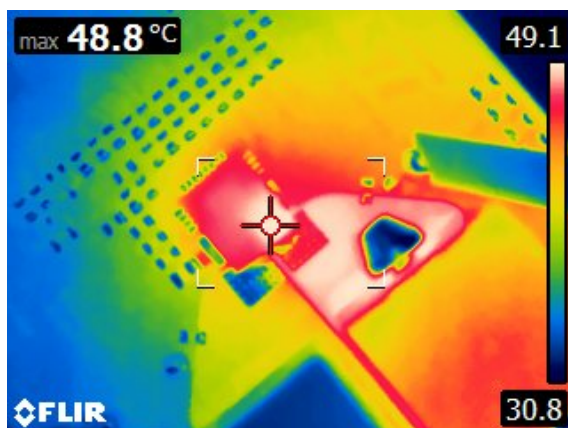




Figure 67. VNF9D1M5Q @ 26 A (OUT0) - Thermal image: 300 secs



VNF9D1M5Q @ 26 A (OUT1)

Figure 68. VNF9D1M5Q @ 26 A (OUT1) - Thermal image: 5 secs

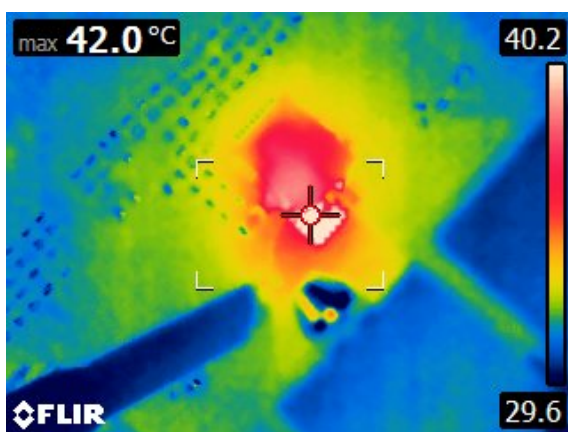


Figure 69. VNF9D1M5Q @ 26 A (OUT1) - Thermal image: 160 secs

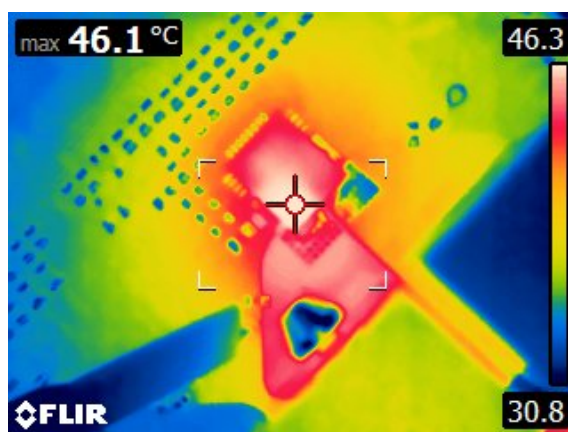
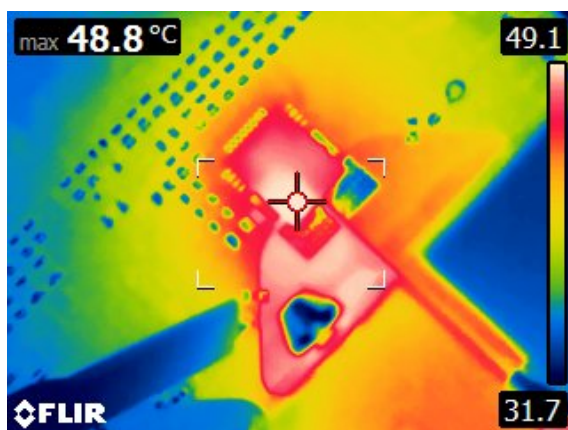


Figure 70. VNF9D1M5Q @ 26 A (OUT1) - Thermal image: 300 secs



VNF9D1M5Q @ 25 A (OUT2)

With a 26 A electronic load, the output latches off before completing the measurement at 300 seconds. For this reason, the test was performed at 25 A.

Figure 71. VNF9D1M5Q @ 25 A (OUT2) - Thermal image: 5 secs

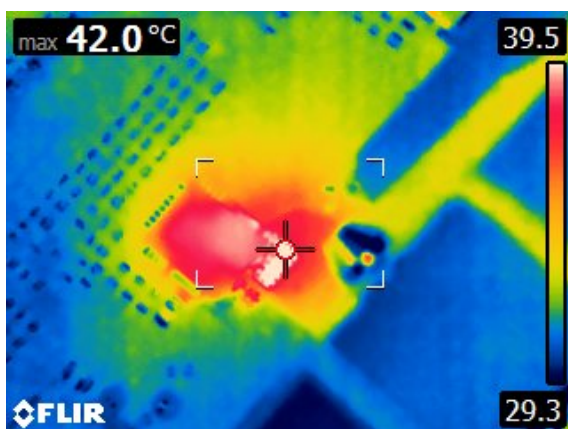


Figure 72. VNF9D1M5Q @ 25 A (OUT2) - Thermal image: 160 secs

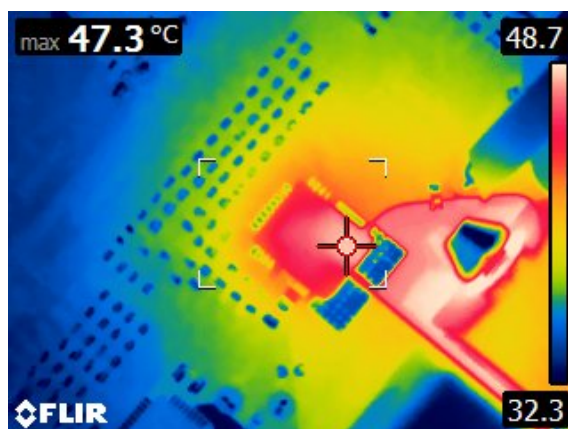
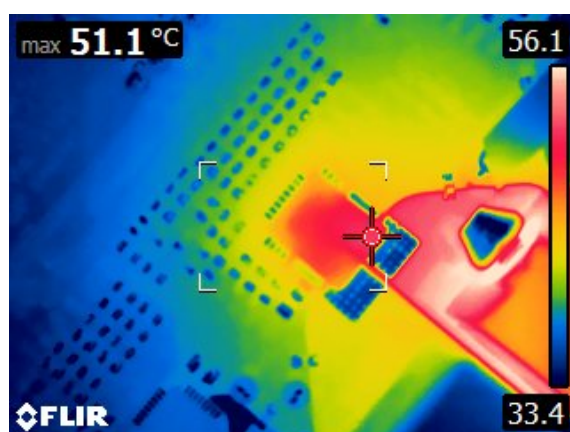


Figure 73. VNF9D1M5Q @ 25 A (OUT2) - Thermal image: 300 secs



#### VNF9D1M5Q @ 25 A (OUT3)

With a 26 A electronic load, the output latches off before completing the measurement at 300 seconds. For this reason, the test was performed at 25 A.

Figure 74. VNF9D1M5Q @ 25 A (OUT3) - Thermal image: 5 secs

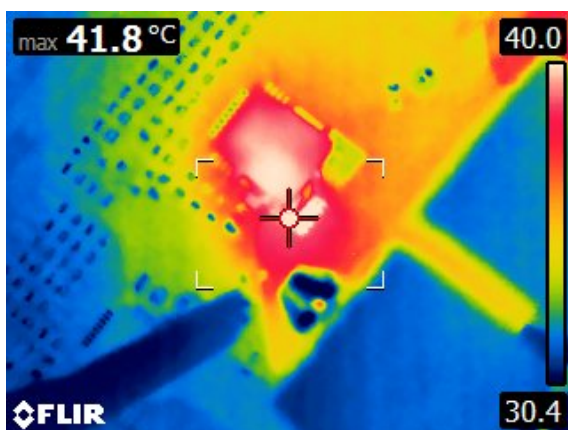


Figure 75. VNF9D1M5Q @ 25 A (OUT3) - Thermal image: 160 secs

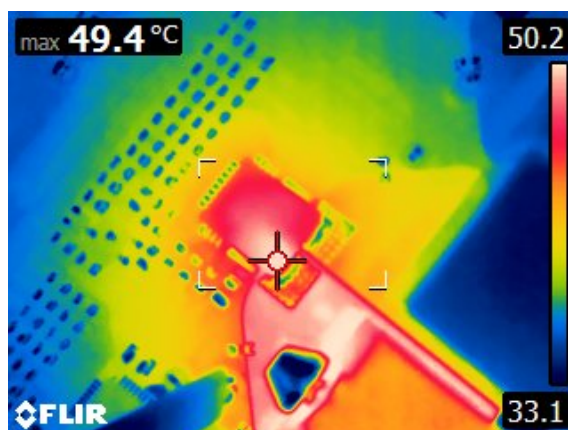
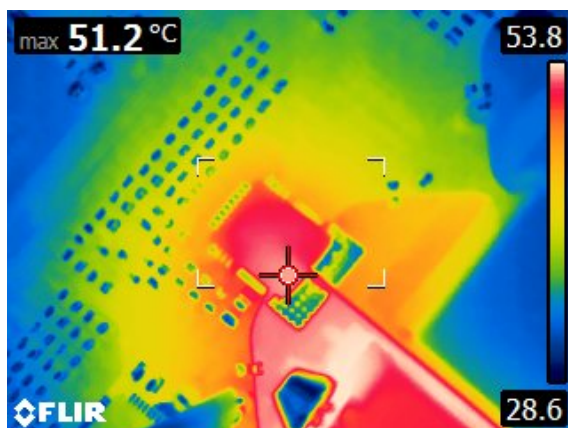


Figure 76. VNF9D1M5Q @ 25 A (OUT3) - Thermal image: 300 secs



VNF9D5SF @ 14 A (OUT4)

Figure 77. VNF9D5SF @ 14 A (OUT4) - Thermal image: 5 secs

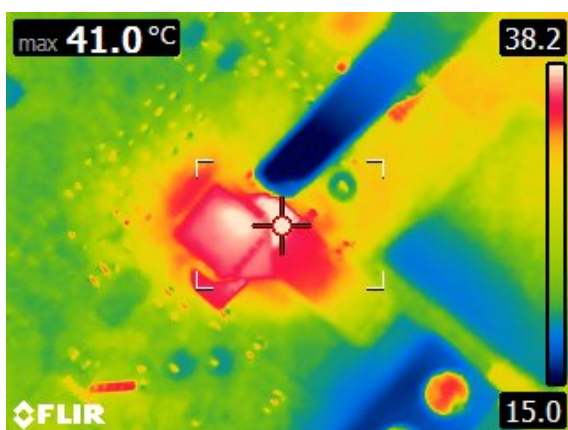


Figure 78. VNF9D5SF @ 14 A (OUT4) - Thermal image: 160 secs

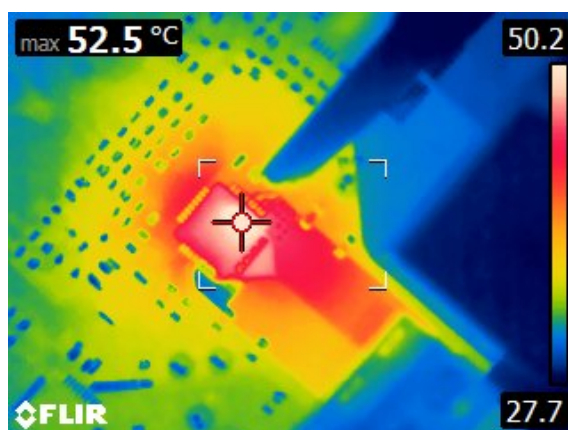
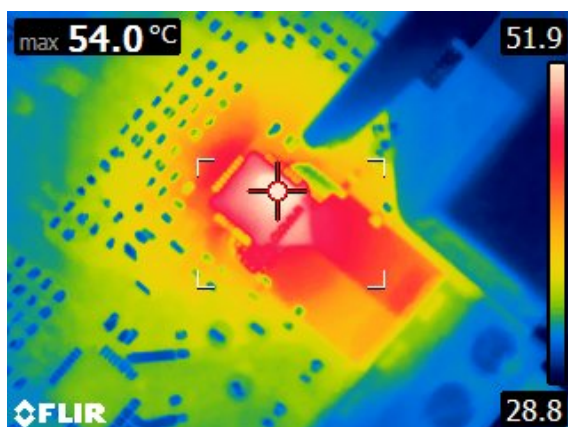


Figure 79. VNF9D5SF @ 14 A (OUT4) - Thermal image: 300 secs





VNF9D5SF @ 14 A (OUT5)

Figure 80. VNF9D5SF @ 14 A (OUT5) - Thermal image: 5 secs

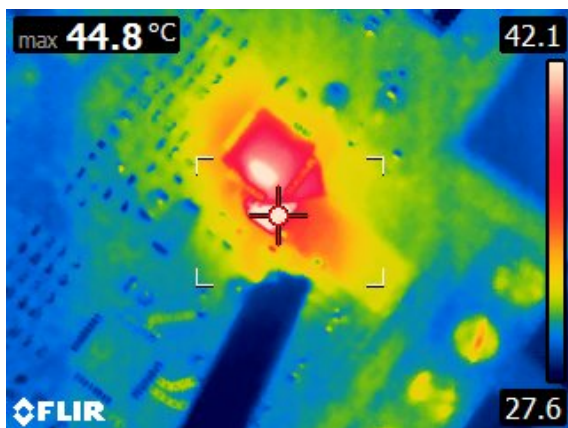


Figure 81. VNF9D5SF @ 14 A (OUT5) - Thermal image: 160 secs

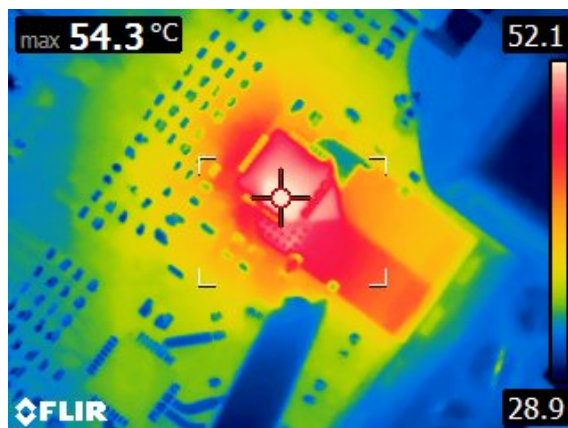
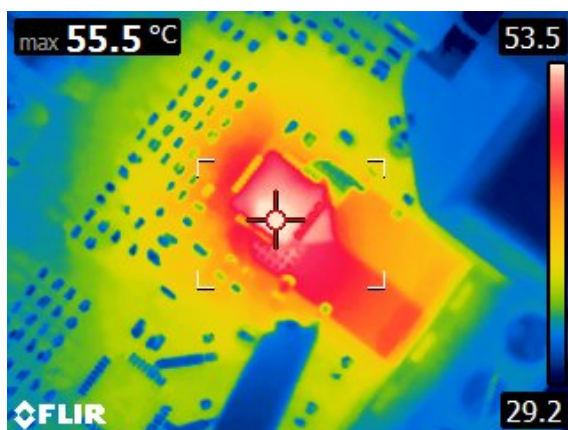


Figure 82. VNF9D5SF @ 14 A (OUT5) - Thermal image: 300 secs



VNF9Q20SF @ 5 A (OUT6)

Figure 83. VNF9Q20SF @ 5 A (OUT6) - Thermal image: 5 secs

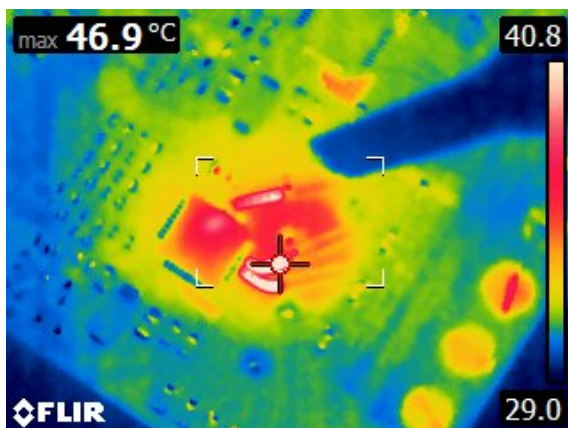


Figure 84. VNF9Q20SF @ 5 A (OUT6) - Thermal image: 160 secs

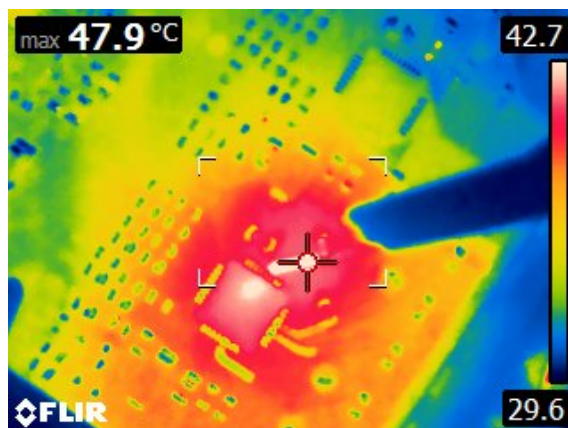
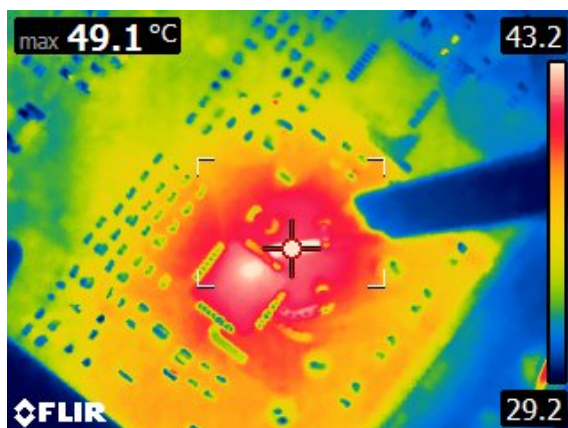




Figure 85. VNF9Q20SF @ 5 A (OUT6) - Thermal image: 300 secs



VNF9Q20SF @ 5 A (OUT7)

Figure 86. VNF9Q20SF @ 5 A (OUT7) - Thermal image: 5 secs

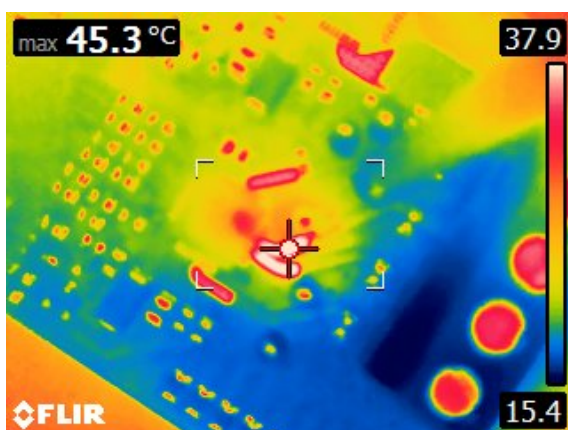


Figure 87. VNF9Q20SF @ 5 A (OUT7) - Thermal image: 160 secs

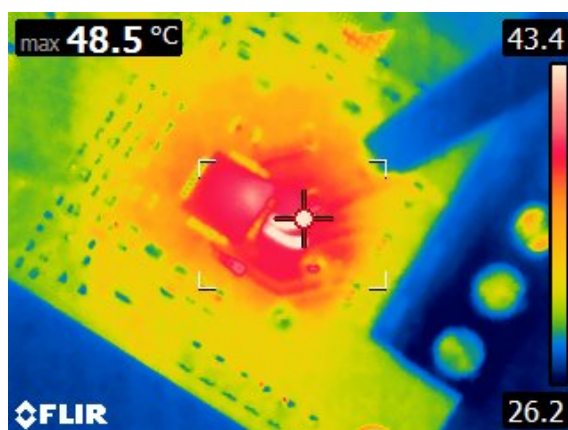
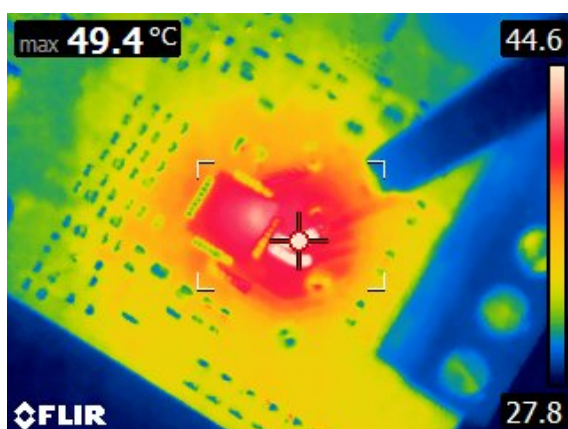


Figure 88. VNF9Q20SF @ 5 A (OUT7) - Thermal image: 300 secs



VNF9Q20SF @ 5 A (OUT8)

Figure 89. VNF9Q20SF @ 5 A (OUT8) - Thermal image: 5 secs

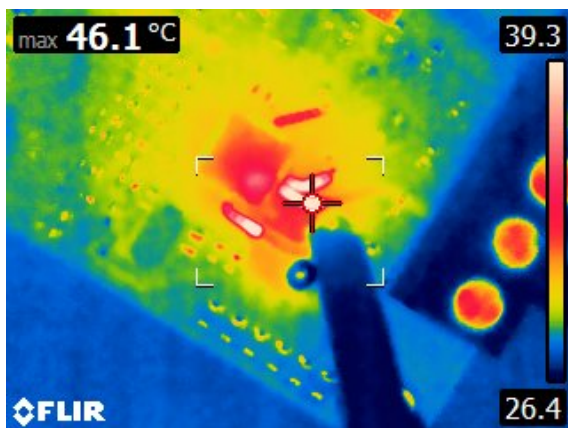


Figure 90. VNF9Q20SF @ 5 A (OUT8) - Thermal image: 160 secs

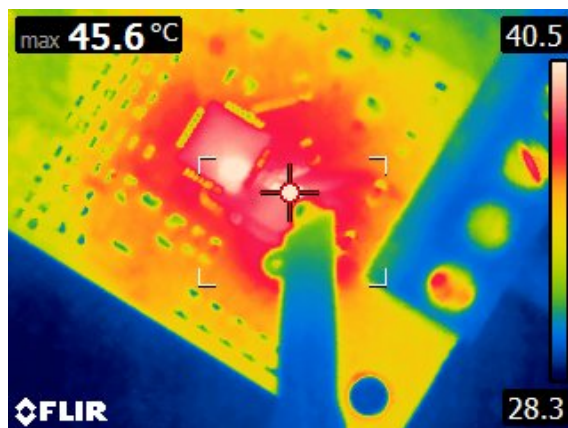
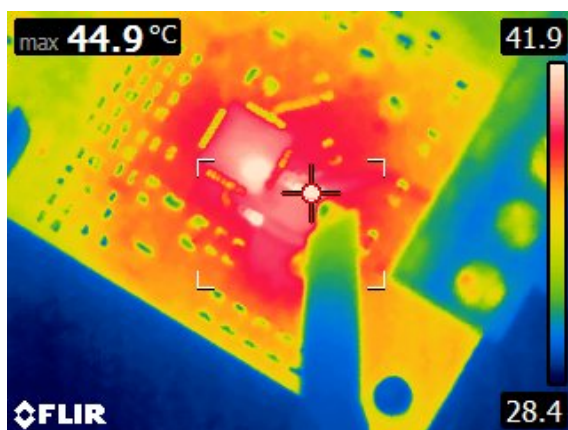


Figure 91. VNF9Q20SF @ 5 A (OUT8) - Thermal image: 300 secs



VNF9Q20SF @ 5 A (OUT9)

Figure 92. VNF9Q20SF @ 5 A (OUT9) - Thermal image: 5 secs

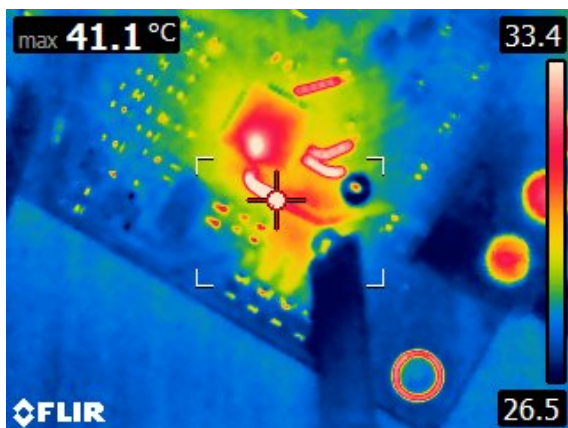


Figure 93. VNF9Q20SF @ 5 A (OUT9) - Thermal image: 160 secs

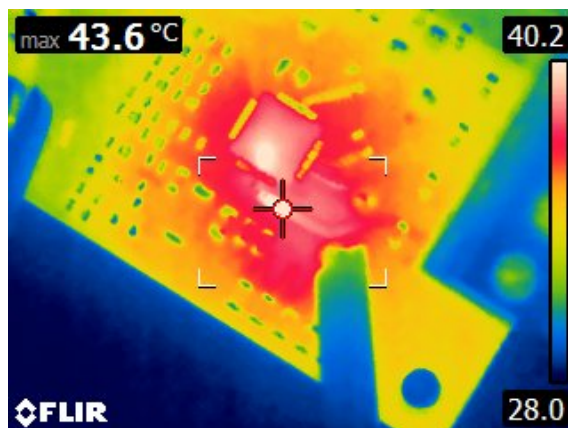
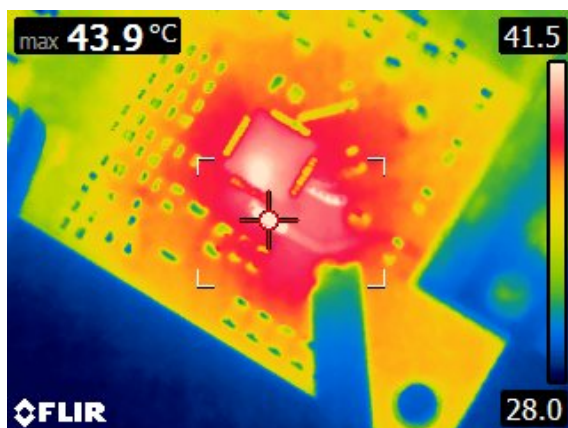


Figure 94. VNF9Q20SF @ 5 A (OUT9) - Thermal image: 300 secs



### 3.5.3 Combination of devices for maximum current

In this test, a combination of loads was used to pass a great current through AEK-POW-PDUMINI. The following configuration was used:

- OUT0 & OUT1: 24 A + 24 A = 48 A.
- OUT2 & OUT3: 24 A + 24 A = 48 A.
- OUT4 & OUT5: 14 A + 14 A = 28 A.

The current was run through the outputs for about 300 secs. For each output, at least three temperatures were acquired at different times: at 5 secs (startup), after 160 secs and after 300 secs (measurement stopped).

#### VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1)

Figure 95. VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1)  
- Thermal image: 5 secs

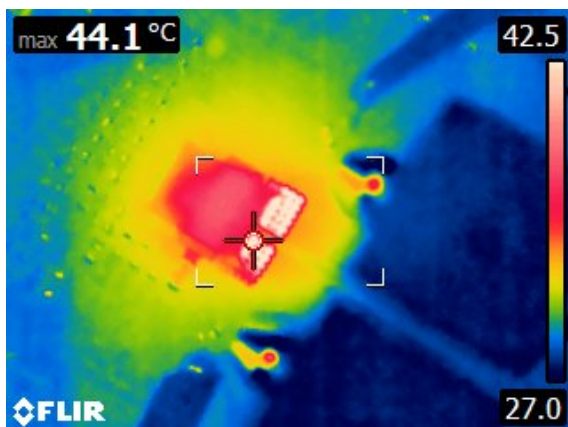


Figure 96. VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1)  
- Thermal image: 160 secs

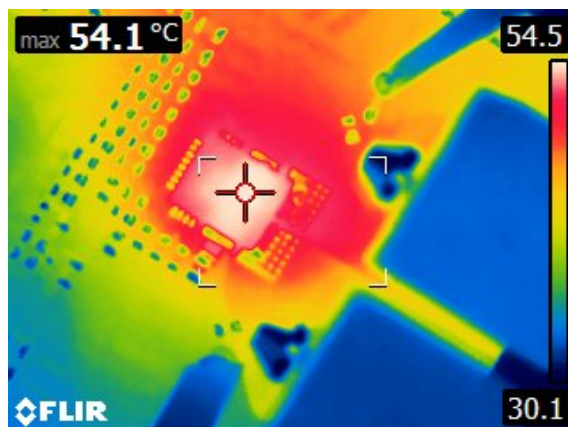
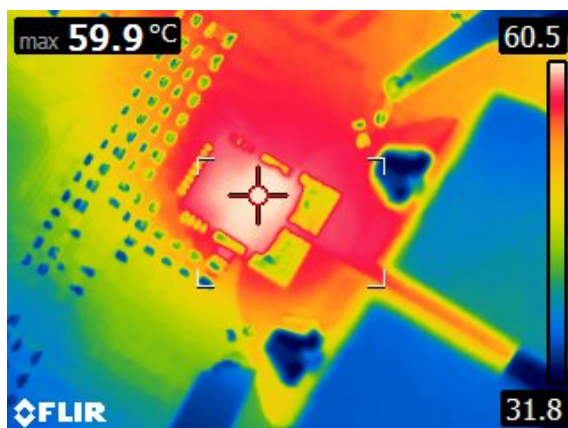




Figure 97. VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1) - Thermal image: 300 secs



VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3)

Figure 98. VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3) - Thermal image: 5 secs

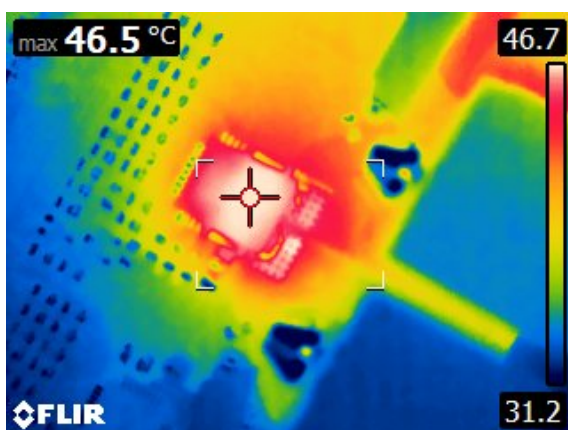


Figure 99. VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3) - Thermal image: 160 secs

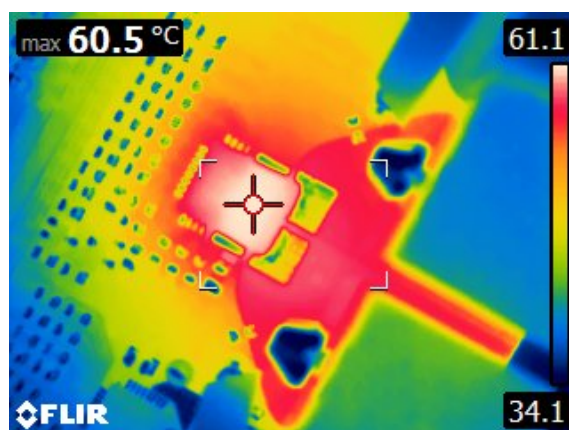
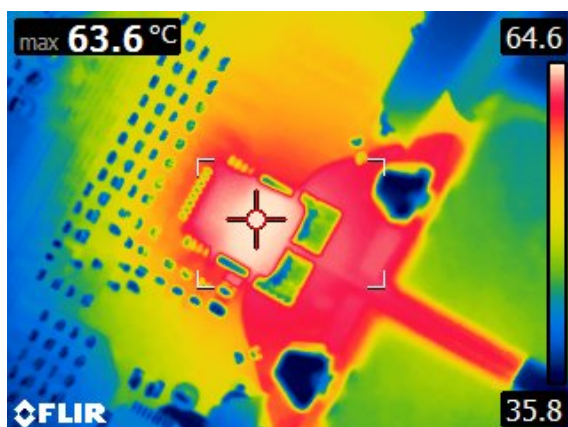


Figure 100. VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3) - Thermal image: 300 secs





VNF9D5SF @ 14 A+14 A (OUT4+OUT5)

Figure 101. VNF9D5SF @ 14 A+14 A (OUT4+OUT5) - Thermal image: 5 secs

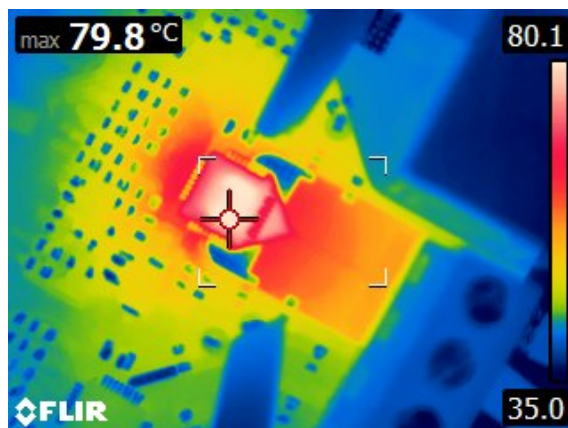
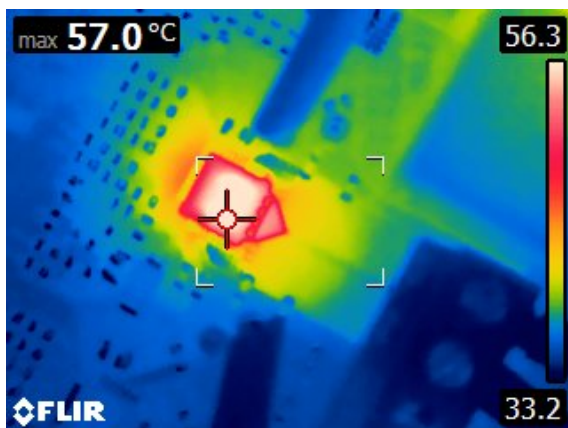
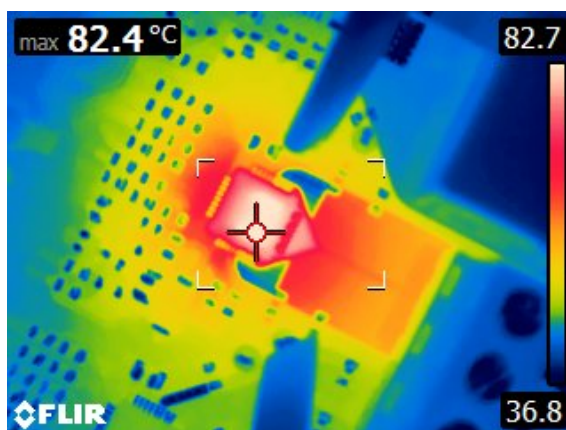


Figure 103. VNF9D5SF @ 14 A+14 A (OUT4+OUT5) - Thermal image: 300 secs



### 3.5.4 Results discussion

In single-channel configurations, the temperature range reached by each device is considered acceptable with a maximum value of 52°C for VNF9D1M5Q and 55.5°C for VNF9D5SF.

In the combined channel configuration, the results show a temperature increase of up to 82.4°C with two VNF9D5SF channels active.

## 3.6 Independent operation of multichannel devices

### 3.6.1 Test conditions

1. Power supply: PSI10080-120 (14 V @ 80 A).
2. Electronic load: 63640-80-80 and 63630-80-60 in single mode (both CCH mode).
3. MXO5 oscilloscope: voltage passive and two current probes (RT-ZC20B).

### 3.6.2 VNF9D1M5Q (device 0)

Channels OUT0 and OUT1 were used. Both outputs are active and configured as follows:

- OUT0,  $I_{nom} = 21$  A and  $t_{nom} = 1$  s.
- OUT1,  $I_{nom} = 16$  A and  $t_{nom} = 172$  s.

Both were loaded with the following load currents: OUT0 @ 20 A and OUT1 @ 15 A.

OUT0 load current was increased to 25 A, causing an overload that triggered the  $I^2t$  threshold.

## Results

As can be seen on the scope capture, the OUT0 latched off (@ 66.62 ms) and the OUT1 remained active.

Figure 104. VNF9D1M5Q (dev0), OUT0 and OUT1 independent channel operation



Figure 105. VNF9D1M5Q (dev0), OUT0 and OUT1 independent channel operation (details of latch off time)



### 3.6.3 VNF9D1M5Q (device 1)

Channels OUT2 and OUT3 were used. Both outputs are active and configured as follows:

- OUT2,  $I_{nom} = 21\text{ A}$  and  $t_{nom} = 1\text{ s}$ .
- OUT3,  $I_{nom} = 16\text{ A}$  and  $t_{nom} = 172\text{ s}$ .

Both were loaded with the following load currents: OUT2 @ 20 A and OUT3 @ 15 A.

OUT2 load current was increased to 25 A, causing an overload that triggered the  $I^2t$  threshold.

## Results

As can be seen on the oscilloscope capture, the OUT2 latched off (@ 61.43 ms) and the OUT3 remained active.

Figure 106. VNF9D1M5Q (dev1), OUT2 and OUT3 independent channel operation



Figure 107. VNF9D1M5Q (dev1), OUT2 and OUT3 independent channel operation (details of latch off time)



### 3.6.4

#### VNF9D5SF

Channels OUT4 and OUT5 were used. Both outputs are active and configured as follows:

Both were loaded with the following load currents: OUT4 @ 3 A and OUT5 @ 8 A.

OUT4 load current was increased to 22 A, causing an overload that triggered the  $I^2t$  threshold.

## Results

As can be seen on the oscilloscope capture, the OUT4 latched off (@ 86.97 ms) and the OUT5 remained active.

**Figure 108. VNF9D5SF, OUT4 and OUT5 independent channel operation**

**Figure 109. VNF9D5SF, OUT4 and OUT5 independent channel operation (details of latch off time)**


### 3.6.5

#### VNF9Q20SF

Channels OUT0 and OUT1 were used. Both outputs are active and configured as follows:

Channels OUT6 and OUT7 were used. Both outputs are active and configured as follows:

- OUT6,  $I_{nom} = 6\text{ A}$  and  $t_{nom} = 300\text{ s}$ .
- OUT7,  $I_{nom} = 6\text{ A}$  and  $t_{nom} = 300\text{ s}$ .

#### Results

As can be seen on the scope capture, the OUT6 latched off (@ 488.35 ms, not compliant @ 585 ms simulated) and the OUT7 remained active.



**Figure 110. VNF9Q20SF, OUT6 and OUT7 independent channel operation**

**Figure 111. VNF9Q20SF, OUT6 and OUT7 independent channel operation (details of latch off time)**


### 3.7 P-channel companion chip functionality

These tests are meant to show the functionality of the **L99SP08** P-channel companion chip together with the integrated **STi<sup>2</sup>Fuse** devices. The output of the companion chip was activated with a constant current load of 50 mA (electronic load). Then, the load current was changed to 2 A and the reaction of the companion chip and the connected **STi<sup>2</sup>Fuse** device was recorded.

For each measurement, the following signals were acquired:  $V_{ENX}$ ,  $V_{OUTx}$ ,  $V_{bat}$ ,  $V_{DIAG\_SP08}$ , and  $I_{OUTx}$ .

#### 3.7.1 Test conditions

1. The firmware "PC58ECxx\_RLA AEK\_POW\_PDUMINI" dated 11/11/2025 was used.
2. Power supply: PSI10080-120 (14 V @ 80 A).
3. eLoad: 63640-80-80 in parallel with 63630-80-60 (CCH mode).
4. MXO5 oscilloscope: voltage passive and current probes (RT-ZC20B).

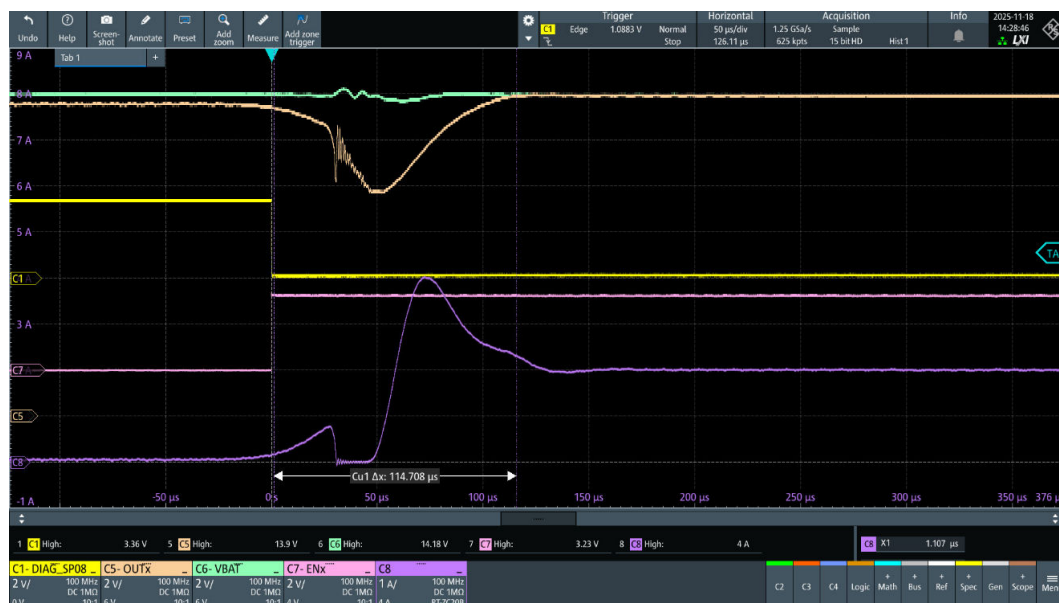
### 3.7.2 VNF9D1M5Q

OUT0 has been selected for this measurement in two different modes: one in Normal mode with L99SP08 activated with a constant current (50 mA) and another with the module in Parking mode.

## Results

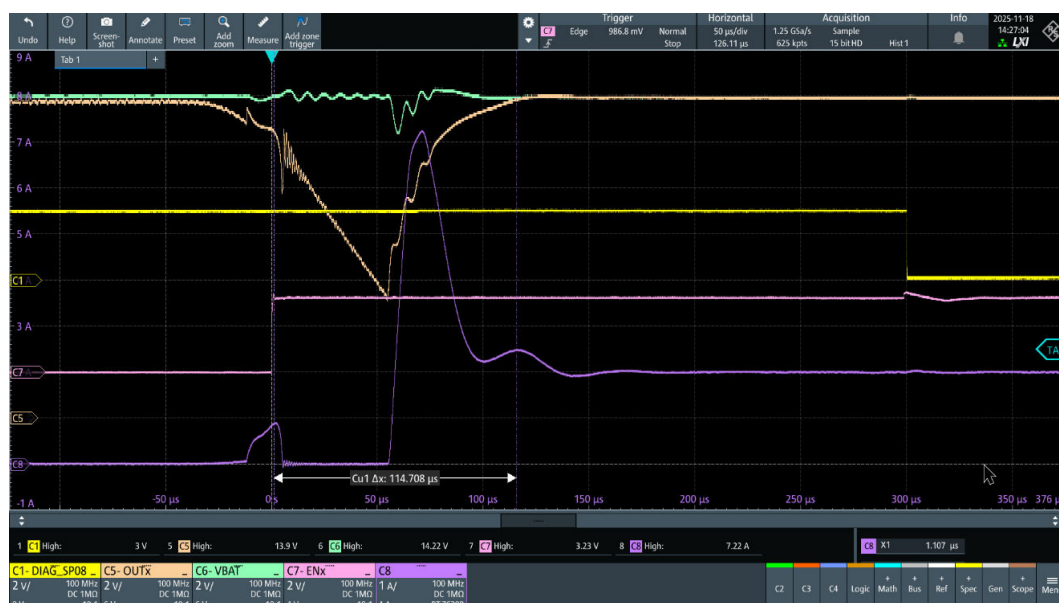
In Normal mode, the current overload measured during the output switching from L99SP08 to STI<sup>2</sup>Fuse was approximately 4 A.

**Figure 112. OUT0, P-channel companion chip activating VNF9D1M5Q device in Normal mode**



In Parking mode, the current overload measured during the output switching from L99SP08 to STi<sup>2</sup>Fuse was approximately 7.5 A.

**Figure 113. OUT0, P-channel companion chip activating VNF9D1M5Q device in Parking mode**

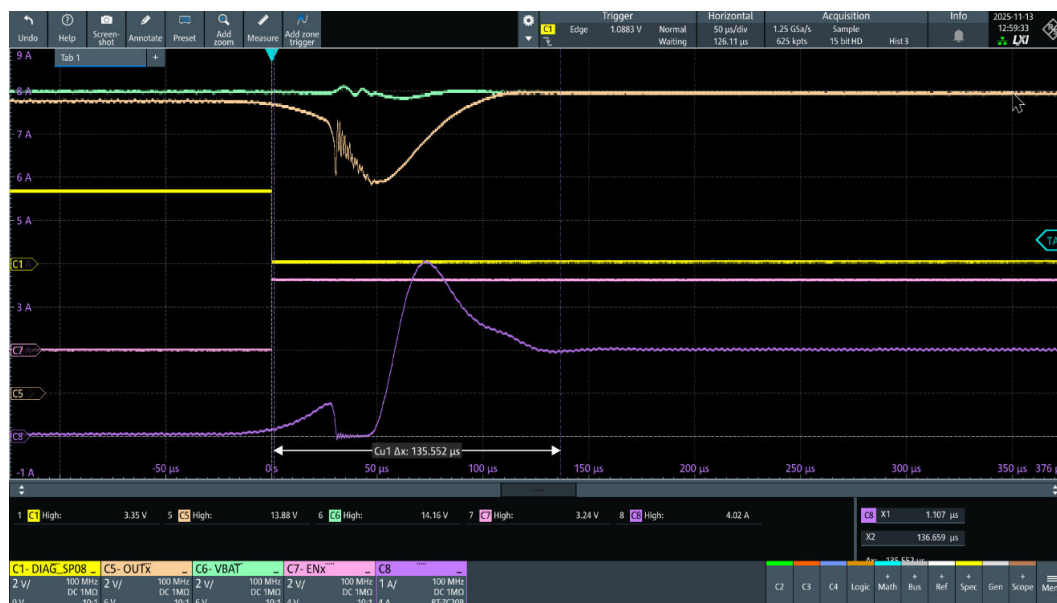


OUT1 has been selected for this measurement only in Normal mode with L99SP08 activated with a constant current (50 mA).

## Results

The current overload measured during the output switching from L99SP08 to ST<sup>2</sup>Fuse was approximately 4 A.

**Figure 114. OUT1, P-channel companion chip activating VNF9D1M5Q device**



OUT2 has been selected for this measurement only in Parking mode for undesired behavior when the channel is activated. In fact, when the electronic load is connected but not active and the channel (by L99SP08) is activated immediately afterwards, a threshold exceedance is detected (that is, DIAG\_SP08 goes low and ENx goes high), even though there is no current consumption (eLoad is off).

## Results

The current overload measured during the output switching from L99SP08 to ST<sup>2</sup>Fuse was approximately 7.5 A.

**Figure 115. OUT2, P-channel companion chip activating VNF9D1M5Q device**



OUT3 has been selected for this measurement only in Parking mode for undesired behavior when the channel is activated. In fact, when the electronic load is connected but not active and the channel (by [L99SP08](#)) is activated immediately afterwards, a threshold exceedance is detected (that is, DIAG\_SP08 goes low and ENx goes high), even though there is no current consumption (eLoad is off).

## Results

The current overload measured during the output switching from L99SP08 to STi<sup>2</sup>Fuse was approximately 7.5 A.

**Figure 116. OUT3, P-channel companion chip activating VNF9D1M5Q device**





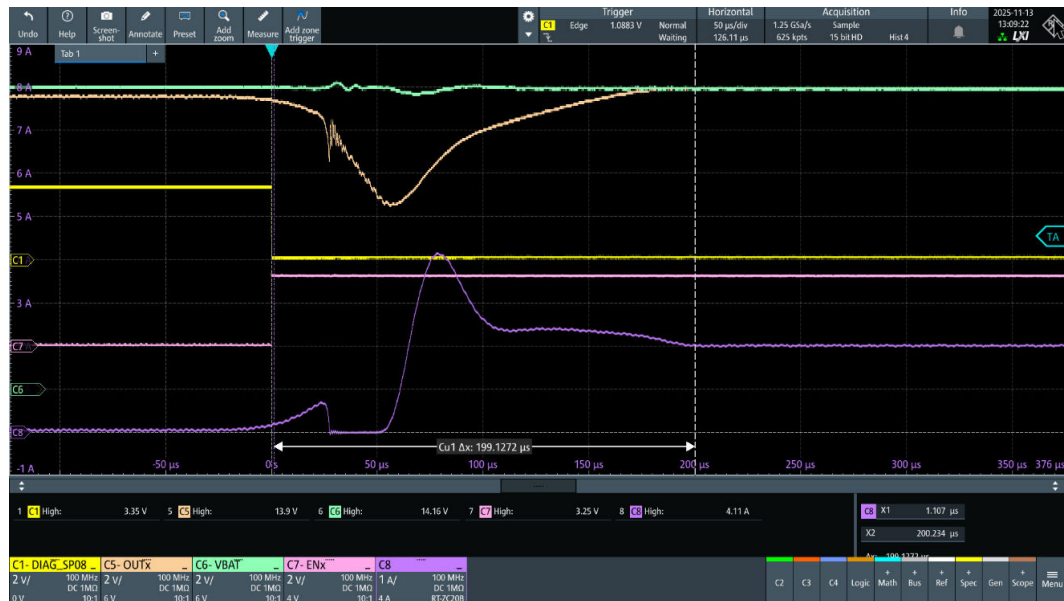
### 3.7.3 VNF9D5SF

OUT4 has been selected for this measurement only in Normal mode with L99SP08 activated with a constant current (50 mA).

#### Results

The current overload measured during the output switching from L99SP08 to ST<sup>2</sup>Fuse was approximately 4 A.

Figure 117. OUT4, P-channel companion chip activating VNF9D5SF device

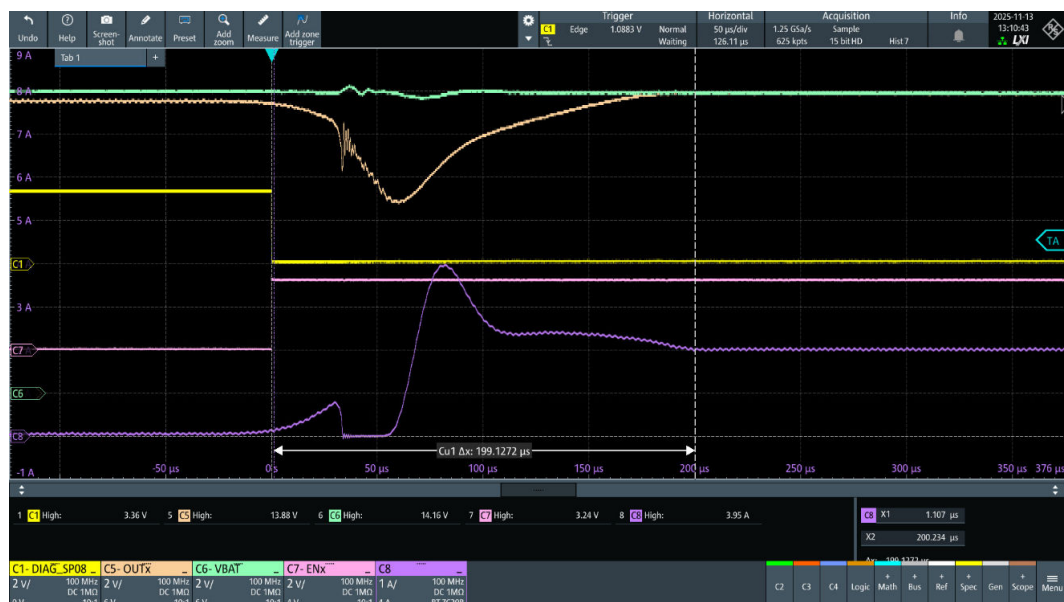


OUT5 has been selected for this measurement only in Normal mode with L99SP08 activated with a constant current (50 mA).

#### Results

The current overload measured during the output switching from L99SP08 to ST<sup>2</sup>Fuse was approximately 4 A.

Figure 118. OUT5, P-channel companion chip activating VNF9D5SF device



### 3.7.4 VNF9Q20SF

OUT6 has been selected for this measurement only in Normal mode with L99SP08 activated with a constant current (50 mA).

#### Results

The current overload measured during the output switching from L99SP08 to ST<sup>2</sup>Fuse was approximately 3.4 A.

Figure 119. OUT6, P-channel companion chip activating VNF9Q20SF device



OUT7 has been selected for this measurement only in Normal mode with L99SP08 activated with a constant current (50 mA).

#### Results

The current overload measured during the output switching from L99SP08 to ST<sup>2</sup>Fuse was approximately 3.4 A.

Figure 120. OUT7, P-channel companion chip activating VNF9Q20SF device



**Note:** The current overload measured above is dependent on the electronic load connected to their output.

## 4 Software description

### 4.1 The AutoDevKit ecosystem

The application development employing the [AEK-POW-PDUMINI](#) takes full advantage of the AutoDevKit ecosystem, whose basic components are:

- AutoDevKit Studio IDE ([STSW-AUTODEVKIT](#)).
- OpenOCD programmer and debugger.

AutoDevKit Studio is an integrated development environment (IDE) based on Eclipse. Its aim is to maximize productivity of embedded applications based on SPC5 Power® architecture 32-bit microcontrollers with a single tool for evaluation, development, design, and production.

AutoDevKit Studio includes an application wizard to simplify project creation and configuration. It automatically creates component dependencies and generates support files.

Other advantages of AutoDevKit Studio include:

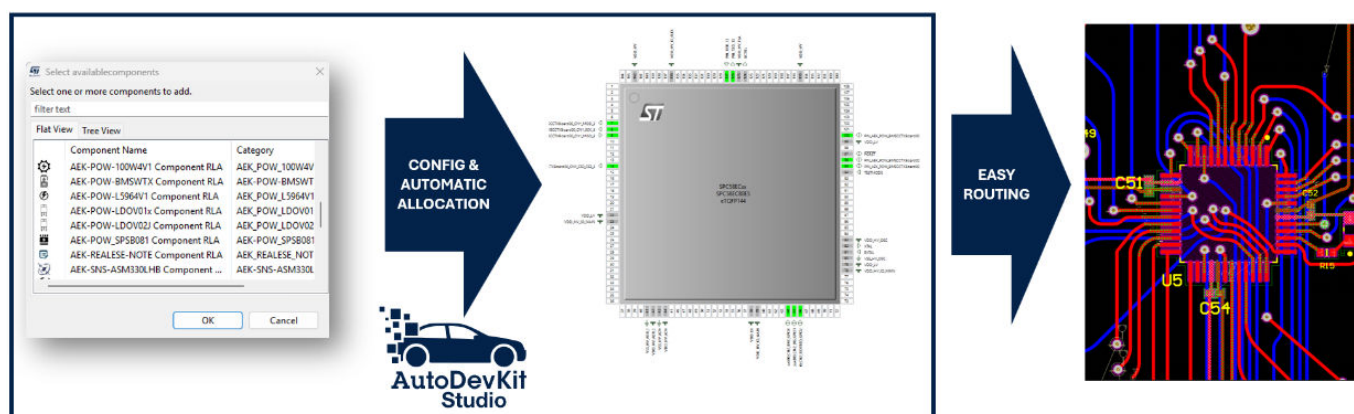
- The ability to integrate other software products from the Eclipse standard marketplace.
- The availability of a free license GCC GNU C compiler component.
- The possibility to support industry-standard compilers.
- Support of multicore microcontrollers.
- A PinMap editor to verify MCU pin configuration.
- A set of demos available in source code.

AutoDevKit is a particularly useful ecosystem for designing a power distribution unit. Starting from this IDE, it is possible to easily generate a new project, importing various components, such as PMICs, e-fuses (for example, [VNF9D1M5Q](#), [VNF9Q20SF](#), [VNF9D5SF](#), and [L99SPS08](#)), needed for load management and protection.

The tool allows automatic configuration and allocation of MCU pins and automatically generates software drivers for the selected components, drastically reducing development time and minimizing the risk of errors.

Starting from the PinMap editor, it is possible to build schematic diagrams and PCBs, ensuring that the relevant pins are correctly assigned and compatible with the project logic. This way, AutoDevKit enables fast transition from idea to prototyping, ensuring flexibility and reliability in designing advanced power distribution systems.

**Figure 121. Example of fast prototyping using AutoDevKit**



The platform also adapts to modular and scalable PDU projects, such as multiboard power distribution unit architectures. This is possible by using [STEVAL-PDUBV1](#) as primary power distribution unit, or a dedicated zonal controller, depending upon the topology, able to control various [AEK-POW-PDUMINI](#) secondary PDU units. A third option is to use a generic MCU board (for example, [AEK-MCU-C4MLIT3](#)) to control a set of remote satellite [AEK-POW-PDUMINI](#) modules via CAN, 10BASE-T1S, or 100BASE-TX.

AutoDevKit represents a complete solution for developing high-reliability automotive power distribution systems, ready for integration into zonal architectures.

## 4.2 AEK-POW-PDUMINI software stack

The AEK-POW-PDUMINI software stack is available in source code inside AutoDevKit Studio and is freely available for download for ST customers.

**Note:** The software is at engineering-level only, dedicated to lab and bench evaluation.

**Important:** The software is NOT SUITABLE for production NOR for real in-vehicle testing.

The firmware for the power distribution unit (PDU) is designed with a strict two-layer architecture:

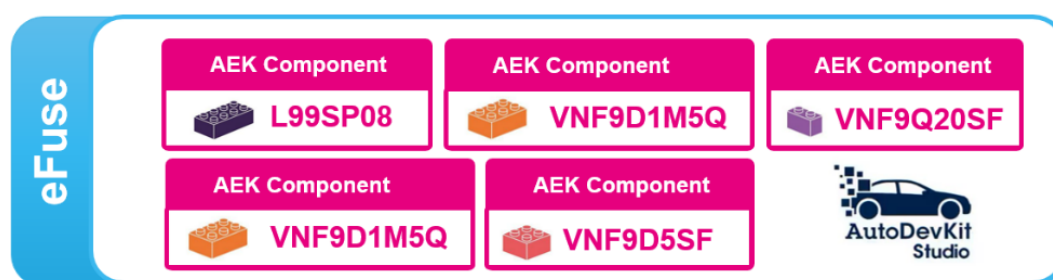
- Driver layer: the driver layer is responsible for all direct hardware access and device-specific logic. It implements the drivers for all the hardware devices used in the system, including:
  - VNF family e-fuse devices (for example, VNF9D1M5Q, VNF9D5SF, VNF9Q20SF).
  - PMIC (SPSB081).
  - L99SP08 e-fuse companion device.
  - SPC58 hardware abstraction layer (HAL), that is, all MCU peripherals and protocols are implemented and simplified APIs, namely low-level drivers, are provided to the upper layers for implementation. The peripherals involved include ADC, CAN, GPIO, PWM, timers (ST), EEPROM, etc.

Figure 122. SPC8 HAL portion



In this layer, for each programmable IC component present on the AEK-POW-PDUMINI board, a specific component driver is developed. The component driver contains the library implementing all the IC functions according to the datasheet specifications, and it is provided with a GUI interface in AutoDevKit Studio to perform user configurations. These configurations are then used to generate C source code for the application. There are dedicated components also for other complex ICs, like the PMICs also requiring a dedicated driver to be programmed via SPI. Inside each component driver, there are device primitives for initialization, device-level register access, device sub-level register access (for example, a specific channel of a multichannel e-fuse), and low-level error management.

Figure 123. Portion of the device layer dedicated to e-fuse devices



- Service layer: the service layer includes all tasks running on the freeRTOS real-time operating system (that is, a dedicated version ported by ST for SPC58 microcontrollers). The application logic tasks are part of this layer, interleaved with other service layer tasks. For implementation simplicity, there is no execution level ID difference between service and application tasks. The other service layer tasks are diagnostics, task management, inter-module communication, and system supervision and monitoring. These tasks use the driver layer APIs to interact with the hardware. Therefore, they never access specific IC registers or hardware peripherals directly. The layered software stack ensures an abstraction from lower to upper layer. This translates to customization and porting high flexibility as far as the layer interface is duly respected.



#### 4.2.1 AEK-POW-PDUMINI firmware basic startup steps

The firmware entry point at code level is the 'main.c' file. From this file, we can derive the basic initial steps performed by the firmware:

- Hardware and driver initialization: through the `componentsInit()` function, all peripherals and low-level drivers (driver layer) are initialized and configured.
- Interrupt enablement: the `irqIsrEnable()` function enables interrupt handling, which is essential for system interactions.
- Service layer initialization: The `PDU_init()` function is the main entry point for the application logic and the heart of the system scheduling strategy.

#### 4.2.2 Driver layer

The driver layer is the foundation of the [AEK-POW-PDUMINI](#) firmware architecture, providing the essential abstraction between the hardware (MCU and peripherals) and the application logic implemented in the service layer. It encapsulates all hardware-specific details and exposes standardized APIs to the upper layers, ensuring modularity, portability, and maintainability.

Based on the layered firmware architecture, the driver layer handles all hardware-specific operations. Therefore, the service layer can focus on application logic and system behavior only.

The software stack structure also ensures easy maintenance: when there are updates at IC level (for example, a new or improved e-fuse device or MCU), only the driver layer is directly impacted, while the service layer and the application logic remain unaffected.

Another key point is that all low-level error handling, protocol compliance checks, and timings are managed within the drivers, ensuring reliable operation for the upper layer.

##### 4.2.2.1 Driver layer implementation

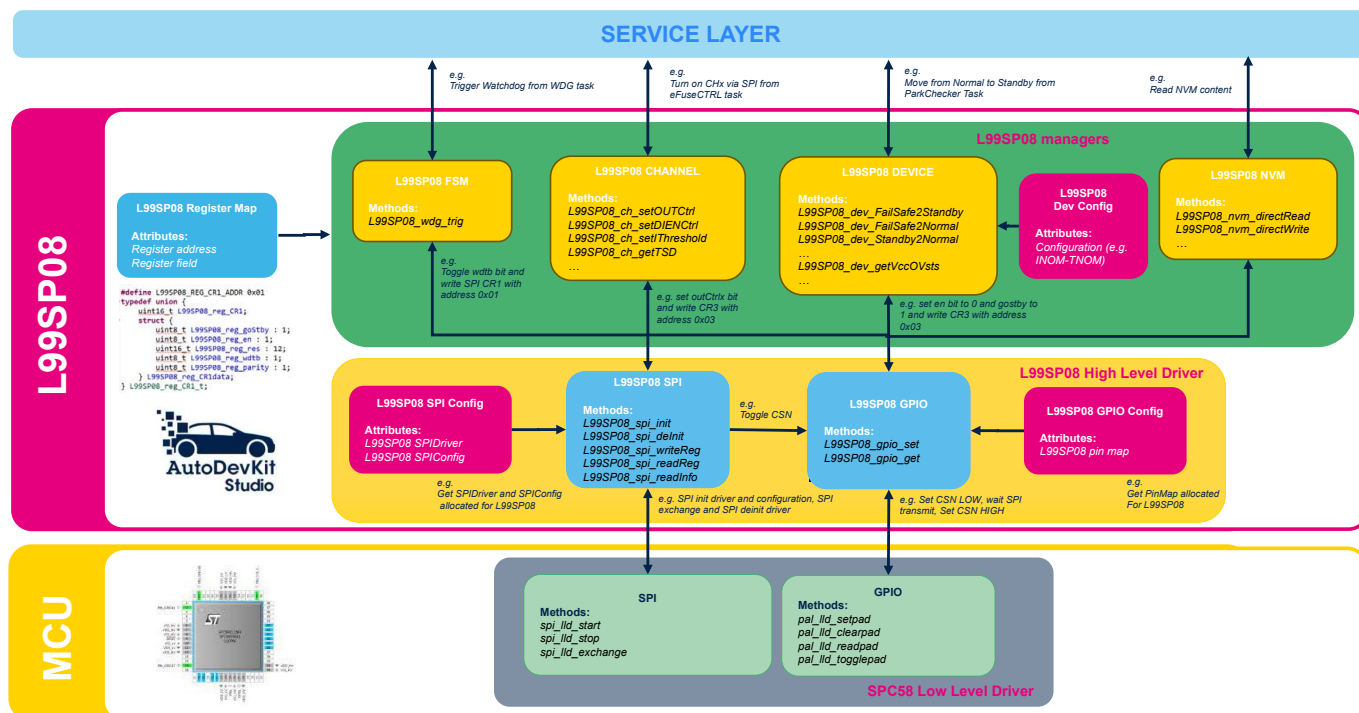
The driver layer consists of a collection of drivers (that is, component driver). Each driver is dedicated to a specific smart IC present on the [AEK-POW-PDUMINI](#) board. All the implemented drivers are designed with a three-level abstraction stack, ensuring that the service layer can interact with the hardware in a device-independent, maintainable, and robust way. In detail, the three-layer stack consists of:

1. Manager layer (top of driver layer): it implements high-level, device-oriented functions such as channel control, device state management, and NVM access without considering how data is physically transmitted to the device. The manager layer simply calls a function to "set a bit" or "write a register," and expects the lower layers to handle the details.
2. High-level driver layer: it bridges the gap between device-level operations and hardware access. In fact, this layer manages the construction of SPI packets, GPIO pin control, and the sequencing of hardware actions required to implement the manager layer requests. The high-level driver does not directly access MCU registers. Instead, it uses the low-level driver APIs for all hardware-related operations.
3. Low-level driver layer (AutoDevKit generated): it provides the direct interface to the MCU hardware peripherals (SPI, GPIO, etc.). These drivers are typically autogenerated by [AutoDevKit Studio](#) and are tightly coupled with the specific MCU and board configuration. They are responsible for register-level access, timing, and protocol compliance.

An example of the implemented three-layer stack for some [L99SP08](#) features is shown below:

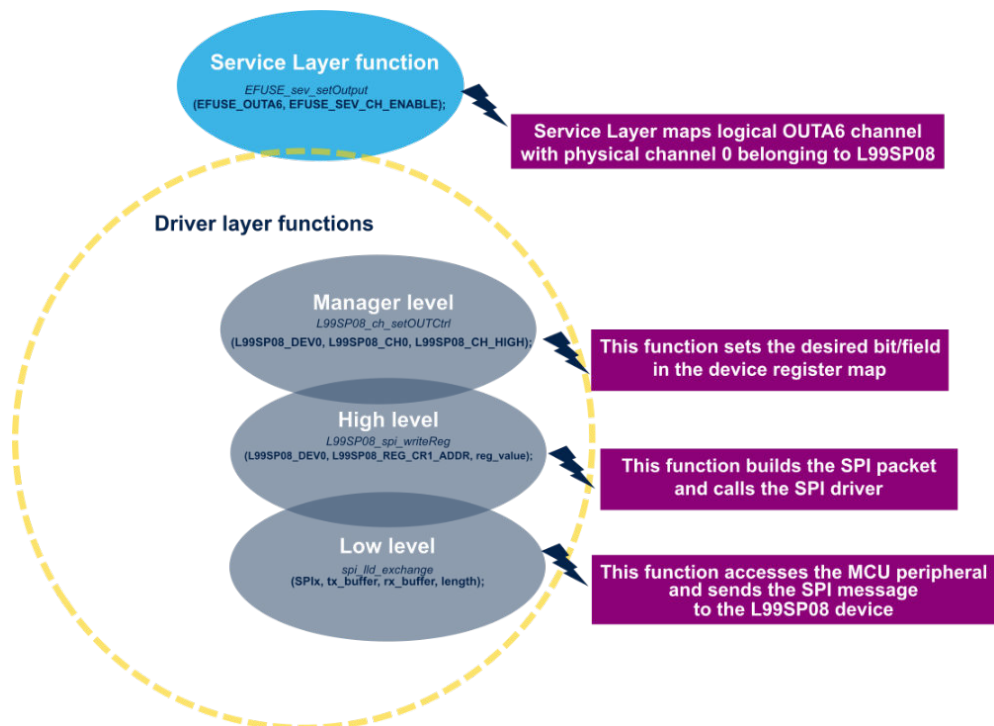
1. Manager layer functions:
  - `L99SP08_ch_setOUTCtrl()`: it sets the output state of a specific channel.
  - `L99SP08_dev_FailSafe2Standby()`: it moves the device from fail-safe to Standby mode.
  - `L99SP08_wdg_trig()`: it triggers the watchdog finite state machine (FSM).
2. High-level driver layer functions:
  - `L99SP08_spi_init()`, `L99SP08_spi_readReg()`, `L99SP08_spi_writeReg()`: they handle SPI initialization and register access (read/write) for [L99SP08](#).
  - `L99SP08_gpio_set()`, `L99SP08_gpio_get()`: they control and read device-specific GPIOs (for example, chip select, DIAG pins).
3. Low-Level driver layer (AutoDevKit generated) functions:
  - `spi_lld_start()`, `spi_lld_stop()`, `spi_lld_exchange()`: they directly access the MCU SPI peripheral.
  - `pal_lld_setpad()`, `pal_lld_clearpad()`, `pal_lld_readpad()`: they directly access the MCU GPIOs.

Figure 124. Three-layer stack for L99SP08 in driver layer



The following figure shows a complete nested function call from a task running in the service layer to the activation of the MCU peripherals. For this example, we consider the operation to turn **L99SP08** channel 0 on. The result of the operation (success or error) is returned to the service layer, which can then update system status, log the event, or notify other modules. This interaction model shows that the service layer can control and monitor all hardware resources in a device-independent way, while the driver Layer ensures safe, robust, and efficient access to the underlying MCU and e-fuse devices.

Figure 125. Nested function calls from service layer to MCU peripheral for L99SP08 Channel 0 activation

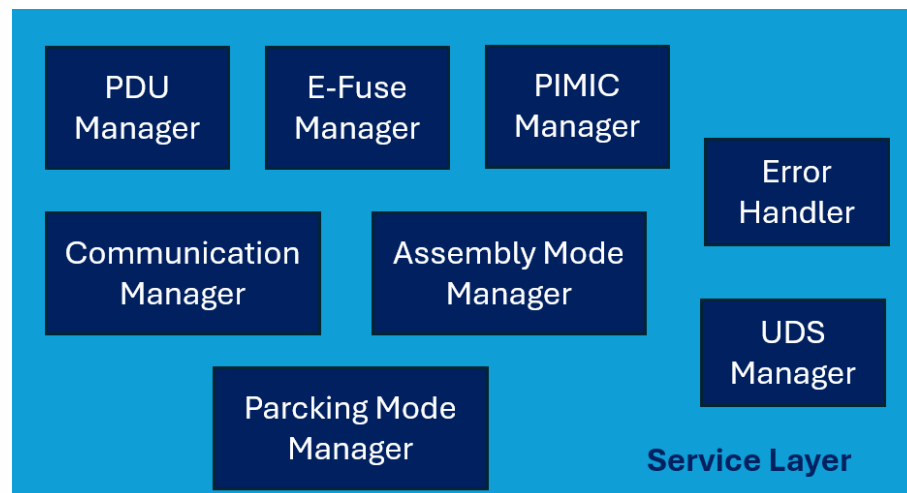


### 4.2.3 Service layer

The service layer consists of the following task collection running on the freeRTOS operating system:

- PDU\_mng: it serves as the central coordinator of the service layer in the PDU firmware.
- EFUSE\_mng: it is responsible for configuring, controlling, and monitoring all e-fuse channels.
- PMIC\_mng: it manages all onboard PMIC ICs.
- COMM\_mng: it manages all communication operations of [AEK-POW-PDUMINI](#).
- ASS\_mng: it is responsible for the configuration of the e-fuses while in Assembly mode.
- PARK\_mng: it manages the transition of the system into Parking mode.
- ERR\_mng: it detects, stores, and reports all errors.
- UDS\_mng: it implements a simplified version of the ISO14229 diagnostics standard.

**Figure 126. Service layer key components**



#### 4.2.3.1 PDU manager task (PDU\_mng)

The PDUB\_mng module serves as the central coordinator of the service layer in the firmware. Its main responsibilities are to initialize all application-level services, configure and start the periodic scheduling mechanism, and provide the main entry point for the application logic after hardware and driver initialization.

As seen in the basic startup section, the PDUB\_init() function is the last to be invoked. The key role is to sequentially initialize all the service layer tasks by invoking their respective functions:

- ASS\_sev\_initTask().
- PMIC\_sev\_initTask().
- EFUSE\_sev\_initTask().
- COMM\_sev\_initTask().
- PARK\_sev\_initTask().
- ERR\_sev\_initTask().
- UDS\_sev\_initTask().

Each routine creates a dedicated freeRTOS task and allocates the necessary synchronization of resources (semaphores, queues, etc.). The periodic activation of these tasks is not managed by using the native freeRTOS scheduling primitives (such as vTaskDelay or vTaskDelayUntil). For this purpose, instead, the firmware leverages a hardware timer-based dispatcher, based on a hardware timer configured to generate periodic interrupts.

Every time it is invoked (that is, every 10 msec), the interrupt handler (callback) of this timer, named PDUB\_stm10\_clbk(), increments a global counter. This global counter value is in turn used to activate tasks according to their required specific intervals (for example, every 10 ms, 40 ms, or 100 ms) by releasing semaphores or sending notifications. Each task remains blocked, waiting for its semaphore/notification, and is executed only when explicitly activated by the dispatcher.

This approach is among the best industrial practices in various time-critical applications like automotive, railway, and medical fields, where safety and predictability are essential requirements. It reinforces temporal determinism by having task activation uniquely controlled by the hardware timer, thus ensuring that periodic operations always occur at the same time, with no jitter accumulation. This centralized control facilitates verification, validation, and temporal certification of the system, thus reducing risk of race conditions, deadlocks, or starvation.

While freeRTOS native scheduling (using `vTaskDelay` or `vTaskDelayUntil`) is more flexible and easier to implement, it does not ensure the same level of determinism and synchronization between tasks. With tasks of different priorities or variable computing loads, delays (jitters) can accumulate, which is unacceptable in safety-critical systems.

#### 4.2.3.2 **eFuse manager task (eFUSE\_mng)**

The eFuse module is a central component of the firmware, responsible for the configuration, control, monitoring, and diagnostics of all e-fuse channels in the system. It provides a unified interface for managing multiple e-fuse devices, ensuring safe power distribution, protection against electrical faults, and support for advanced diagnostics. The key features include:

- Channel control: it enables or disables individual e-fuse channels based on system logic or external commands.
- Parameter monitoring: it continuously monitors voltage, current, and temperature for each channel, providing real-time protection and diagnostics.
- Fault detection and handling: it detects overcurrent, overtemperature, undervoltage, and other fault conditions and executes appropriate recovery and/or protection actions.
- Configuration management: it initializes and configures all e-fuse devices at startup according to predefined parameters and safety thresholds.
- Advanced diagnostics: it supports a wide range of diagnostic and status queries, including support for UDS diagnostic services.

##### 4.2.3.2.1 **Asynchronous queue usage in the e-fuse manager task**

All interactions with the eFuse module are performed using asynchronous message queues. There are two queues: one for requests, and one for responses. Service layer modules like COMM, UDS, PARK, or diagnostic communicate with the eFuse module by placing structured request messages into the dedicated request queue. Each message specifies the operation to be performed (for example, enable/disable a channel, read voltage/current/temperature, enter standby, perform diagnostics) and includes all necessary parameters. The eFuse module extracts and processes the request queue and, after processing a request, it places the corresponding response message into a response queue. The requesting module retrieves the result, either by waiting for the response or by polling the queue.

The eFuse module includes a dedicated freeRTOS task (EFUSECtrlTask) that continuously monitors the request queue, processes incoming requests in a thread-safe manner, interacts with the appropriate hardware drivers, and posts responses to the response queue.

This implementation ensures thread safety and decoupling between modules, enables nonblocking, responsive operations, supports scalability and easy extension with robust error handling and diagnostics.

A typical usage pattern is shown in [Figure 127](#). The requesting task fills out a request structure with the desired operation and parameters. This request is added to the e-fuse request queue. The e-fuse control task (EFUSECtrlTask) processes the request, interacts with the hardware drivers, and places the result in the response queue, while the requesting task waits for response retrieval.



**Figure 127. Example of channel enablement request**

```
// Example: Request to enable channel EFUSE_OUTA6
EFUSE_sev_msgReq_t req;
EFUSE_sev_msgRes_t res;

req.EFUSE_sev_id = EFUSE_SEV_SET_OUTPUT;
req.EFUSE_sev_channel = EFUSE_OUTA6;
req.EFUSE_sev_data8 = EFUSE_SEV_CH_ENABLE;
req.TaskHandle = xTaskGetCurrentTaskHandle(); // So the response can be notified back

EFUSE_sev_addReqQueue(&req);

// Wait for the response (blocking)
if (EFUSE_sev_getResQueue(&res) == pdTRUE) {
    if (res.EFUSE_error == EFUSE_OK) {
        // Output enabled successfully
    } else {
        // Handle error
    }
}
```

#### 4.2.3.2.2 Channel abstraction in the e-fuse manager task

Each logical e-fuse channel is represented by an enum type (`efuse_sev_channel_t`) internally mapped to the specific device and hardware channel. This enum type structure contains the outputs from OUT0 to OUT9. OUT0 to OUT7 correspond to the **L99SP08** channels, while OUT8 and OUT9 correspond to the **VNFDQ20SF** channels. Channels OUT0 to OUT3 are connected to **L99SP08** and to **VNF9D1M5Q**; channels OUT4 and OUT5 are connected to both **L99SP08** and **VNF9D5SF**; channels OUT6 and OUT7 are connected to **L99SP08**, as well as to two channels of **VNF9Q20SF**. This mapping is handled in the implementation of functions like `EFUSE_sev_setOutput()`, and `EFUSE_sev_getVoltage()` by using switch/case statements or configuration tables. With this abstraction, the rest of the system can refer to channels in a device-independent way. The eFuse module handles all the logic to determine which driver function to call for each channel. Therefore, when adding new devices or changing hardware mappings, changes are required only within the eFuse module, leaving the rest of the applications unchanged.

#### 4.2.4 eFuse module implemented functions

The main functions implemented in the eFuse module are listed below, grouped by purpose.

**Figure 128. eFuse module functions**

| E-fuse module function                                                                                      | Description                                                                                 |
|-------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| <code>void EFUSE_sev_initTask(void);</code>                                                                 | It initializes the e-fuse tasks, queues, and semaphores.                                    |
| <code>void EFUSE_sev_runTask(void);</code>                                                                  | It triggers the main e-fuse task (typically from an ISR or a scheduler).                    |
| <code>void EFUSE_sev_addReqQueue(EFUSE_sev_msgReq_t *req);</code>                                           | It adds a request to the e-fuse request queue (from other modules).                         |
| <code>void EFUSE_sev_addResQueue(EFUSE_sev_msgRes_t *res);</code>                                           | It adds a response to the e-fuse response queue.                                            |
| <code>BaseType_t EFUSE_sev_getResQueue(EFUSE_sev_msgRes_t *res);</code>                                     | It retrieves a response from the response queue (blocking until available).                 |
| <code>EFUSE_error_t EFUSE_sev_setOutput(EFUSE_sev_channel_t channel, EFUSE_sev_channelSts_t status);</code> | It enables or disables a specific e-fuse channel.                                           |
| <code>EFUSE_error_t EFUSE_sev_setStandby(void);</code>                                                      | It sets all e-fuse devices and channels into standby mode.                                  |
| <code>void EFUSE_sev_enable(void);</code>                                                                   | It enables the e-fuse system (e.g., clears standby pins).                                   |
| <code>void EFUSE_sev_disable(void);</code>                                                                  | It disables the e-fuse system (e.g., sets standby pins).                                    |
| <code>EFUSE_error_t EFUSE_sev_Config(void);</code>                                                          | It configures all e-fuse devices at startup, setting thresholds and operational parameters. |
| <code>void EFUSE_sev_setECUsts(EFUSE_sev_ECUsts_t status);</code>                                           | It sets the ECU state for e-fuse management.                                                |

The following figure lists the functions dedicated to monitoring and diagnostics.

**Figure 129. eFuse module diagnostics and monitoring functions**

| E-fuse module function for monitoring and diagnostics                                         |                                                    | Description                                           |
|-----------------------------------------------------------------------------------------------|----------------------------------------------------|-------------------------------------------------------|
| <code>EFUSE_error_t EFUSE_sev_getVoltage(EFUSE_sev_channel_t channel, float *vout);</code>    |                                                    | It reads the voltage of a channel.                    |
| <code>EFUSE_error_t EFUSE_sev_getCurrent(EFUSE_sev_channel_t channel, float *current);</code> |                                                    | It reads the current of a channel                     |
| <code>EFUSE_error_t EFUSE_sev_getTemp(EFUSE_sev_channel_t channel, float *temp);</code>       |                                                    | It reads the temperature of a channel.                |
| <code>EFUSE_error_t EFUSE_sev_getOutput(EFUSE_sev_channel_t channel, uint8_t *status);</code> |                                                    | It gets the output status of a channel.               |
| Advanced diagnostics and status queries                                                       |                                                    |                                                       |
| <code>EFUSE_error_t EFUSE_sev_getOVC()</code>                                                 | <code>EFUSE_error_t EFUSE_sev_getVDSHSR()</code>   | <code>EFUSE_error_t EFUSE_sev_getGSBN()</code>        |
| <code>EFUSE_error_t EFUSE_sev_getOVT()</code>                                                 | <code>EFUSE_error_t EFUSE_sev_getIPEAKLSR()</code> | <code>EFUSE_error_t EFUSE_sev_getPTWFAIL()</code>     |
| <code>EFUSE_error_t EFUSE_sev_getUV()</code>                                                  | <code>EFUSE_error_t EFUSE_sev_getCHLOFFSR()</code> | <code>EFUSE_error_t EFUSE_sev_getPVDSMAXFAIL()</code> |
| <code>EFUSE_error_t EFUSE_sev_getI2T()</code>                                                 | <code>EFUSE_error_t EFUSE_sev_getTW2()</code>      | <code>EFUSE_error_t EFUSE_sev_getPOSCFAIL()</code>    |
| <code>EFUSE_error_t EFUSE_sev_getDIENSR()</code>                                              | <code>EFUSE_error_t EFUSE_sev_getITST()</code>     | <code>EFUSE_error_t EFUSE_sev_getPVCCOVFAIL()</code>  |
| <code>EFUSE_error_t EFUSE_sev_getCAPCSR()</code>                                              | <code>EFUSE_error_t EFUSE_sev_getTCTN()</code>     | <code>EFUSE_error_t EFUSE_sev_getPVCCUVFAIL()</code>  |
| <code>EFUSE_error_t EFUSE_sev_getPTSDFAIL()</code>                                            | <code>EFUSE_error_t EFUSE_sev_getOTSR()</code>     | <code>EFUSE_error_t EFUSE_sev_getPTSDFAIL()</code>    |

#### 4.2.5 eFuse module interaction with other modules

The eFuse module main interactions with other modules in the service layer are the following ones:

- COMM: it receives and processes commands from the communication module (for example, set output, get status).
- UDS: it provides diagnostic data and supports UDS service requests for e-fuse channels.
- PARK: it executes standby or shutdown procedures upon request.
- ERR: it reports fault conditions and supports error logging and recovery.

#### 4.2.6 e-fuse device configuration

The configuration of all e-fuse devices and their channels is managed through dedicated configuration structures, as defined in the `EFUSE_sev_cfg.h` file and referenced in the `EFUSE_sev_mng.h` file. This section explains the meaning and usage of each parameter.

**Table 4. Device-dependent channel parameters**

| Device    | Field                  | Type                    | Description                                                                              |
|-----------|------------------------|-------------------------|------------------------------------------------------------------------------------------|
| L99SP08   | L99SP08_ch_ithr0..7    | L99SP08_ch_ithr_t       | Nominal current threshold for each channel (0-7). Sets the current limit for protection. |
| L99SP08   | L99SP08_ch_ccm0..7     | L99SP08_ch_sts_t        | Continuous current mode (CCM) status for each channel (0-7). Enables/disables CCM mode.  |
| VNF9D1M5Q | VNF9D1M5Q_inom0_level  | VNF9D1M5Q_inom_level_t  | Nominal current for channel 0                                                            |
| VNF9D1M5Q | VNF9D1M5Q_inom1_level  | VNF9D1M5Q_inom_level_t  | Nominal current for channel 1                                                            |
| VNF9D1M5Q | VNF9D1M5Q_ipeak0_level | VNF9D1M5Q_ipeak_level_t | Peak current for channel 0                                                               |
| VNF9D1M5Q | VNF9D1M5Q_ipeak1_level | VNF9D1M5Q_ipeak_level_t | Peak current for channel 1                                                               |
| VNF9D1M5Q | VNF9D1M5Q_tnom0_level  | VNF9D1M5Q_tnom_level_t  | Nominal time for channel 0                                                               |
| VNF9D1M5Q | VNF9D1M5Q_tnom1_level  | VNF9D1M5Q_tnom_level_t  | Nominal time for channel 1                                                               |
| VNF9D1M5Q | VNF9D1M5Q_dev_tempTh   | VNF9D1M5Q_dev_tempTh_t  | Device temperature threshold                                                             |
| VNF9D1M5Q | EFUSE_sev_cfg_sts      | EFUSE_sev_cfg_sts_t     | Configuration status                                                                     |
| VNF9D1M5Q | VNF9D1M5Q_iLimOff0     | Float                   | LCC current threshold for channel 0                                                      |
| VNF9D1M5Q | VNF9D1M5Q_iLimOff1     | Float                   | LCC current threshold for channel 1                                                      |
| VNF9D5SF  | VNF9D5SF_ch_inom0      | VNF9D5SF_ch_inom_t      | Nominal current for channel 0                                                            |

| Device    | Field                     | Type                       | Description                                                                               |
|-----------|---------------------------|----------------------------|-------------------------------------------------------------------------------------------|
| VNF9D5SF  | VNF9D5SF_ch_inom1         | VNF9D5SF_ch_inom_t         | Nominal current for channel 1                                                             |
| VNF9D5SF  | VNF9D5SF_ch_tnom0         | VNF9D5SF_ch_tnom_t         | Nominal time for channel 0                                                                |
| VNF9D5SF  | VNF9D5SF_ch_tnom1         | VNF9D5SF_ch_tnom_t         | Nominal time for channel 1                                                                |
| VNF9D5SF  | EFUSE_sev_cfg_sts         | EFUSE_sev_cfg_sts_t        | Configuration status                                                                      |
| VNF9D5SF  | VNF9D5SF_iLimOff0         | Float                      | LCC current threshold for channel 0                                                       |
| VNF9D5SF  | VNF9D5SF_iLimOff1         | Float                      | LCC current threshold for channel 1                                                       |
| VNF9D5SF  | VNF9D5SF_ch_ishutoff0     | VNF9D5SF_ch_ishutOffThr_t  | Shut-off time for channel 0 (time after which the channel is turned off in case of fault) |
| VNF9D5SF  | VNF9D5SF_ch_ishutoff1     | VNF9D5SF_ch_ishutOffThr_t  | Shutoff time for channel 1                                                                |
| VNF9D5SF  | VNF9D5SF_ch_imlLatch0     | VNF9D5SF_ch_imlLatch_t     | Autorestart for channel 0 (enables/disables automatic restart after fault)                |
| VNF9D5SF  | VNF9D5SF_ch_imlLatch1     | VNF9D5SF_ch_imlLatch_t     | Autorestart for channel 1                                                                 |
| VNF9Q20SF | VNF9Q20SF_ch_inom0..3     | VNF9Q20SF_ch_inom_t        | Nominal current for each channel (0-3)                                                    |
| VNF9Q20SF | VNF9Q20SF_ch_tnom0..3     | VNF9Q20SF_ch_tnom_t        | Nominal time for each channel (0-3)                                                       |
| VNF9Q20SF | EFUSE_sev_cfg_sts         | EFUSE_sev_cfg_sts_t        | Configuration status                                                                      |
| VNF9Q20SF | VNF9Q20SF_iLimOff0..3     | Float                      | LCC current threshold for each channel (0-3)                                              |
| VNF9Q20SF | VNF9D5SF_ch_ishutoff0..3  | VNF9Q20SF_ch_ishutOffThr_t | Shutoff time for each channel (0-3)                                                       |
| VNF9Q20SF | VNF9Q20SF_ch_imlLatch0..3 | VNF9Q20SF_ch_imlLatch_t    | Autorestart for each channel (0-3)                                                        |

On top of all above specific parameters, there are two global parameters:

- **EFUSE\_CURRENT\_TH**: It is of type float. It represents the current threshold (in Amperes) below which a channel is considered off (used for standby and safety checks).
- **EFUSE\_VOLTAGE\_TH**: It is of type float. It represents the voltage threshold (in Volts) for standby and protection of logic.

#### 4.2.7

#### e-fuse channel transition from Active to Standby mode

The following steps show how to safely transition all e-fuse devices and channels into Standby mode, ensuring the system is in a safe, low-power state before shutdown or during parking.

- Step 1.** Checks for active loads and faults. It iterates over all relevant channels (from OUT0 to OUT9). For each channel, it checks if the channel is enabled (DIENSR) and, if so, reads the current. If any channel has a current above a defined threshold (EFUSE\_CURRENT\_TH), the function returns an error, preventing unsafe standby.
- Step 2.** Checks for device faults. For each L99SP08 device and channel, it checks for VDSMAX or TSD (thermal shutdown) faults. If any such fault is present, it returns an error.
- Step 3.** Device transitions to standby. For L99SP08 devices, it calls L99SP08\_dev\_Normal2Standby() to enter standby mode. Finally, it calls EFUSE\_sev\_disable() to set the standby pins for the system.
- Step 4.** Return: If all checks pass and all devices are transitioned successfully, it returns EFUSE\_OK.

**Important:** *This function is comprehensive and defensive: it does not allow Standby mode if any unsafe condition is detected on any of the available devices (L99SP08, etc.). This behavior is fundamental to avoid entering standby in a potentially critical condition.*

#### 4.2.8 PMIC\_mng of the service layer

The PMIC module is a key component of the [AEK-POW-PDUMINI](#) firmware, responsible for the configuration, control, monitoring, and diagnostics of all power management ICs (PMICs) in the system. It manages [SPSB081](#) PMIC, ensuring safe power sequencing, voltage regulation, and support for advanced diagnostics and low-power modes. The key implemented functions include:

- Power mode control: it commands each PMIC device to enter standby, low-power, or active modes, supporting safe shutdown, wake up, and energy management.
- Diagnostic queries: it retrieves status and error flags (overvoltage, undervoltage, watchdog, BIST, power-up, ground loss, thermal warning, etc.) from each PMIC.
- Configuration management: it initializes and configures all PMIC devices at startup according to predefined parameters and safety thresholds (for example, voltage regulator settings and wake up sources).
- Advanced diagnostics: it supports a wide range of diagnostic and status queries, including support for UDS diagnostic services and error logging.

All interactions with the PMIC module are performed using asynchronous message queues:

- Request queue: other service layer modules, such as PARK, UDS, COMM, or diagnostics, communicate with the PMIC module by placing structured request messages into a dedicated request queue. Each message specifies the operation to be performed (for example, enter standby, get overvoltage status, configure wake up sources) and includes all necessary parameters.
- Response queue: after processing a request, the PMIC module places the corresponding response message into a response queue. The requesting module retrieves the result, either by waiting for the response or by polling the queue.

The PMIC module includes a dedicated freeRTOS task (PMICCtrlTask) that continuously monitors the request queue, processes incoming requests in a thread-safe manner, interacts with the appropriate hardware drivers, and posts responses to the response queue. This ensures thread safety and decoupling between modules; it enables non-blocking responsive operations and remains open for scalability and easy extension. The following image shows an example of the code:

**Figure 130. Example of communication with PMIC module**

```
// Example: Request to put SPSB081 into VBAT standby mode
PMIC_sev_msgReq_t req;
PMIC_sev_msgRes_t res;

req.PMIC_sev_id = PMIC_SEV_VBATSTANDBY_SPSB081;
req.TaskHandle = xTaskGetCurrentTaskHandle(); // For response notification

PMIC_sev_addReqQueue(&req);

// Wait for the response (blocking)
if (PMIC_sev_getResQueue(&res) == pdTRUE) {
    // Check result or handle error
}
```

In the previous code, the requesting task fills out a request structure with the desired operation and parameters. The request is added to the PMIC request queue. The PMIC control task (PMICCtrlTask) processes the request, interacts with the hardware drivers, and places the result in the response queue where the requesting task waits for and retrieves the response.

Operations are identified by an enum (PMIC\_sev\_id\_t), and the control task uses the switch/case logic to call the appropriate driver function for the selected device and operation. The key benefit is that the rest of the system can request power management or diagnostics without knowing which PMIC device is present. This also facilitates support for new PMICs or new operations.

Typical interactions with the PMIC include:

- PARK: it requests standby or shutdown operations for safe system power-down.
- UDS: it requests diagnostic information for UDS services.
- COMM: it sends and receives PMIC-related messages over CAN or UART.
- ERR: it may be queried for error status or to support error logging and recovery.



The following figure lists the functions implemented for the PMIC module.

**Figure 131. PMIC module functions**

| PMIC module function                                                  | Description                                                                                                                              |
|-----------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <code>void PMIC_sev_initTask(void);</code>                            | It initializes the PMIC tasks, queues, and semaphores.                                                                                   |
| <code>void PMIC_sev_runTask(void);</code>                             | It triggers the main PMIC task (typically from an ISR or scheduler).                                                                     |
| <code>void PMIC_sev_addReqQueue(PMIC_sev_msgReq_t *req);</code>       | It adds a request to the PMIC request queue (from other modules).                                                                        |
| <code>void PMIC_sev_addResQueue(PMIC_sev_msgRes_t *res);</code>       | It adds a response to the PMIC response queue.                                                                                           |
| <code>BaseType_t PMIC_sev_getResQueue(PMIC_sev_msgRes_t *res);</code> | It retrieves a response from the response queue (blocking until available).                                                              |
| <code>PMIC_sev_id_t (enum);</code>                                    | It enumerates all supported PMIC operations (e.g., standby, low-power mode, get overvoltage, get watchdog error, etc.).                  |
| <code>PMIC_sev_msgReq_t / PMIC_sev_msgRes_t;</code>                   | Structures for request/response messages, including operation ID, data, and task handle.                                                 |
| <code>AEK_POW_SPSB081_Force_VBAT_StandBy</code>                       | Standby & low-power mode commands (device specific).                                                                                     |
| -                                                                     | Diagnostic queries (e.g., get overvoltage, undervoltage, watchdog error, BIST error, power-up error, ground loss, thermal warning, etc.) |
| -                                                                     | Configuration commands (e.g., enable/disable wakeup sources, configure voltage regulators).                                              |

The detailed code behavior below shows the series of operations required to transition all PMIC devices into standby or low-power mode, ensuring the system is in a safe, low-power state before shutdown or during parking:

1. Receives a standby request: the control task receives a request (for example, `PMIC_SEV_VBATSTANDBY_SPSB081`) from the request queue.
2. Calls the device driver: it invokes the appropriate driver function, such as `AEK_POW_SPSB081_Force_VBAT_StandBy(SPSB081_DEV0)`, to command the PMIC to enter Standby mode.
3. Handles device state: it might update the internal state or perform additional configuration to ensure that the device is in the correct mode.
4. Operation result: Places a response in the response queue, indicating success or failure.

As shown, the function is comprehensive and defensive: it does not allow Standby if any unsafe condition is detected.

#### 4.2.9 COMM\_mng of the service layer

The COMM module is a fundamental component of the PDU firmware, responsible for managing all communication between [AEK-POW-PDUMINI](#) and the external devices or systems. It acts as the communication gateway, ensuring reliable message exchange, command dispatching, and support for diagnostics and control. While the current implementation supports the CAN protocol, the module is designed with a high level of abstraction, making it straightforward to extend support to other communication protocols such as UART or Ethernet in the future.

The key implemented functions include:

- External communication gateway: it manages all inbound and outbound communications between the board and external devices (for example, vehicle ECUs, diagnostic tools, or other controllers).
- Message transmission and reception: it handles the message sending and receiving over all supported communication protocols (currently CAN, in future support for UART and Ethernet).
- Command dispatching: it decodes incoming messages and dispatches commands or requests to the appropriate service layer modules (for example, eFuse, PMIC, PARK, and UDS).
- Asynchronous communication: it uses queues and freeRTOS tasks to manage asynchronous, non-blocking message handling.
- Protocol abstraction: it provides a device and protocol-agnostic interface for higher-level modules, abstracting the details of CAN, UART, and Ethernet communication.
- Diagnostics and status reporting: it supports the transmission of diagnostic information, status updates, and error notifications to external systems.

All interactions with the COMM module are performed using asynchronous message queues:

- Transmit queue (Tx): service layer modules (such as eFuse, PMIC, UDS, or diagnostics) send messages to be externally transmitted by placing them into the transmit queue. Each message includes the protocol, destination, and payload.
- Receive queue (Rx): incoming messages from external systems are placed into the received queue by the communication drivers or interrupt callbacks. The COMM module processes these messages and dispatches them to the appropriate internal modules.

The COMM module includes dedicated freeRTOS tasks for transmission (COMMTxTask) and reception (COMMRxTask). These tasks continuously monitor their respective queues, handle message encoding/decoding, and interact with the hardware drivers. This implementation ensures thread safety and decoupling between communication and application logic. The communication operations are non-blocking and responsive, thus supporting scalability and easy extension to new protocols or message types. This is also beneficial for robust error handling and diagnostics.

The COMM module abstracts all supported communication protocols behind a unified API. Each message is identified by a type (COMM\_sev\_msgType\_t) and an ID (COMM\_sev\_msgID\_t). It includes all necessary data and task handles for response notification. The rest of the system can send and receive messages without knowing the details of the underlying protocol, and adding support for new protocols or message types requires changes only within the COMM module. Typical interactions with the COMM module include:

- e-fuse: it receives and dispatches commands to enable/disable channels, read parameters, or perform diagnostics.
- PMIC: it sends and receives power management and diagnostic messages.
- UDS: it handles UDS diagnostic requests and responses.
- PARK: it receives parking/shutdown commands and forwards them as needed.
- ERR: it transmits error notifications and diagnostic information.

The following figure lists the functions implemented for the COMM module.

**Figure 132. COMM module functions**

| COMM module function                                                  | Description                                                                    |
|-----------------------------------------------------------------------|--------------------------------------------------------------------------------|
| <code>void COMM_sev_initTask(void);</code>                            | It initializes the COMM tasks, queues, and semaphores.                         |
| <code>void COMM_sev_addTxQueue(COMM_sev_msg_t *msg);</code>           | It adds a message to the transmit queue (from other modules).                  |
| <code>void COMM_sev_addRxQueue(COMM_sev_msg_t *msg);</code>           | It adds a message to the receive queue (from the hardware driver or ISR).      |
| <code>void COMMRx_sev_runTask(void);</code>                           | It triggers the receive task (typically from an ISR or scheduler).             |
| <code>portTASK_FUNCTION(COMMTxTask, pvParameters);</code>             | FreeRTOS task for handling outgoing messages.                                  |
| <code>portTASK_FUNCTION(COMMRxTask, pvParameters);</code>             | FreeRTOS task for handling incoming messages.                                  |
| <code>void CAN_sev_init(void);</code>                                 | It initializes the CAN driver (if required).                                   |
| <code>void CAN_sev_send(COMM_sev_msg_t msg);</code>                   | It encodes and sends a CAN message.                                            |
| <code>void UART_sev_init(void);</code>                                | It initializes the UART driver (if required).                                  |
| <code>void PDU_CAN_cfg_clbk(uint32_t msgbuf, CANRxFrame crfp);</code> | CAN receive callback, it decodes the message and adds it to the receive queue. |
| <code>void PDU_uartTX_clbk(SerialDriver *sdp);</code>                 | UART transmit callback (placeholder).                                          |
| <code>void PDU_uartRX_clbk(SerialDriver *sdp);</code>                 | UART receive callback (placeholder).                                           |

#### 4.2.10 CAN protocol usage

The AEK-POW-PDUMINI CAN interface is designed for a bus high (BH) network at 125 kbps with the following key parameters:

- Bitrate: 125 kbps (set in the CAN driver configuration).
- Bus type: high-speed CAN (bus high).
- Termination: 120  $\Omega$  resistors at both ends of the bus.
- Physical layer: standard ISO11898-2 transceiver.
- Bus length: up to 500 m (typical for 125 kbps, but please verify your application).
- Node addressing: message routing is based on CAN IDs as defined in the DBC file.

All devices on the CAN network must be configured for 125 kbps and use compatible transceivers. We use a DBC file for all message definitions to ensure interoperability and reduce errors. Commercial CAN tools can be used for diagnostics, commissioning, and automated testing by leveraging the DBC.

The CAN\_db.h file defines the structure, IDs, and bitfields for all CAN messages used in the system. When a CAN message is received, the COMM module uses the macros and definitions from CAN\_db.h to decode the message, extract relevant fields, and convert it into a structured request for the appropriate service layer module. In detail, the CAN driver or ISR receives a CAN frame and calls the callback function PDU\_CAN\_cfg\_clbk(), where the message ID and data fields are parsed using macros from the CAN\_db.h file (for example, CANMSG\_GETFIELD, CANMSG\_CH\_ID). Once the message type and parameters (such as channel number, command, or status) are extracted, a COMM\_sev\_msg\_t structure is filled with the decoded information, specifying the target module, command, and any required data. Then, the decoded message is placed into the COMM receive queue. The COMM RxTask retrieves the message, determines the target module (for example, eFuse), and constructs a corresponding request (for example, EFUSE\_sev\_msgReq\_t). The request is then placed into the target module request queue for processing.

The DBC file is included in the project source directory in AutoDevKit. As previously mentioned, the DBC file can be imported into any major commercial CAN tool (such as vector CANoe, Kvaser CANLIB, Peak PCAN, or similar). Once imported, these tools allow you to easily send, receive, and monitor CAN messages, without having to manually handle bitfields or encoding.

The following table lists the supported CAN messages and IDs.

**Table 5. Supported CAN messages and IDs**

| CAN message name   | CAN ID (Hex) | Purpose/Description                              |
|--------------------|--------------|--------------------------------------------------|
| CANMSG_CH_ID       | 0xAA         | Channel control (enable/disable e-fuse channels) |
| CANMSG_PARKING_ID  | 0xBB         | Parking/standby command                          |
| CANMSG_ASSEMBLY_ID | 0xCC         | Assembly state notification                      |
| CANMSG_UDS_REQ_ID  | 0x7E0        | UDS diagnostic request                           |
| CANMSG_UDS_RES_ID  | 0x7E8        | UDS diagnostic response                          |

**Note:** All message details (IDs, signals, bit positions, allowed values) are defined in the DBC file. You do not need to modify the firmware or manually encode messages. Just use the DBC in your CAN tool.

#### 4.2.11 e-fuse channel activation example via CAN protocol

Supposing that the channel to enable is the EFUSE\_OUT0 the example below shows how to turn-on an e-fuse channel via CAN.

**Step 1.** Receive the CAN frame with ID CANMSG\_CH\_ID.

**Step 2.** Decode the message using the CAN\_db.h.

**Figure 133. Example code of how to turn on an e-fuse channel via CAN**

```
// In PDU_CAN_cfg_clbk()
if (crfp.ID == CANMSG_CH_ID) {
    COMM_sev_msg_t msg;
    msg.COMM_sev_msgType = COMM_SEV_CAN_MSG;
    msg.COMM_sev_msgID = COMM_SEV_SET_OUTPUT;
    msg.EFUSE_sev_channel = CANMSG_GETFIELD(crfp.data32[CANMSG_CH_CHID_FIELD],
CANMSG_CH_CHID_MASK, CANMSG_CH_CHID_SHIFT);
    msg.COMM_sev_data8[0] = CANMSG_GETFIELD(crfp.data32[CANMSG_CH_CHSTS_FIELD],
CANMSG_CH_CHSTS_MASK, CANMSG_CH_CHSTS_SHIFT);
    COMM_sev_addRxQueue(&msg);
}
```

**Step 3.** COMM receive task dispatches the message to eFuse module.

**Figure 134. Example code of message dispatched to the eFuse module**

```
// In COMM RxTask
if (COMM_sev_msgRx.COMM_sev_msgID == COMM_SEV_SET_OUTPUT) {
    EFUSE_sev_msgReq_t req;
    req.EFUSE_sev_id = EFUSE_SEV_SET_OUTPUT;
    req.EFUSE_sev_channel = COMM_sev_msgRx.EFUSE_sev_channel;
    req.EFUSE_sev_data8 = COMM_sev_msgRx.COMM_sev_data8[0]; // 1 = enable, 0 = disable
    req.TaskHandle = COMMTaskRxHandle;
    EFUSE_sev_addReqQueue(&req);
    // Optionally wait for response and handle it
}
```

**Step 4.** The eFuse control task (EFUSECtrlTask) processes the request and enables the specified channel using the appropriate driver.

#### 4.2.12 ASS\_mng of the service layer

The Assembly (ASS) module is responsible for the initial configuration of e-fuse channels during the assembly phase of the ECU. Its main function is to enable or disable specific channels according to a static configuration table, ensuring that the system is correctly set up for initial testing, validation, or production procedures. The key implemented functions are:

- Automated channel activation: the module automatically sends commands to the eFuse module to set the state (enable/disable) of OUT0.
- Verification and feedback: after attempting to configure all channels, the module verifies the actual state of each channel to ensure that the configuration was applied successfully.
- Status reporting: the module provides feedback on the outcome of the assembly procedure, indicating success or failure, and can notify external systems or operators via CAN messages and status LEDs.

Typical interactions with the ASS module include:

- e-fuse: it receives and executes channel configuration commands from ASS during assembly.
- COMM: it sends assembly status notifications (success/failure) via CAN.
- ERR: it might be used to log or handle errors if the assembly procedure fails.
- PARK: it should be used after assembly to safely turn off all loads and put the ECU into a low-power state.

##### 4.2.12.1 Assembly mode detailed operation

###### 4.2.12.1.1 Hardware activation

As described in [Section 2.7](#), Assembly mode is managed via the onboard ASSE\_mode pin. By inserting a jumper in positions 2-3, the assembly mode is activated.

###### 4.2.12.1.2 Mode detection

At startup, the firmware checks the state of the ASSE\_mode pin using the ASS\_sev\_checker() function by reading a GPIO.

If the pin is high, the system state is set to ASS\_ASSEMBLY. Otherwise, the system remains in Normal mode (ASS\_NOT\_ASSEMBLY).

###### 4.2.12.1.3 Output activation sequence

When Assembly mode is active, the ASSTask task performs the following operation sequence:

1. Deactivation of all outputs:  
The software sends commands to turn off all outputs, from EFUSE\_OUT0 to EFUSE\_OUT9.
2. Activation of a specific output:  
The firmware activates only output OUT0 (EFUSE\_OUT0).
3. Status verification:  
After a short delay (100 ms), the software reads back the status of EFUSE\_OUT0 to confirm it is active.



## 4. Feedback and signaling:

- If activation is successful, a CAN message is sent, containing the COMM\_SEV\_SET\_ASSDONE status, and LED1 blinks.
- If activation fails, a CAN message is sent, containing the COMM\_SEV\_SET\_ASSFAIL status, and LED2 is turned on.

**Important:** *Once the assembly procedure is complete and the ECU is operational, it is strongly recommended to turn all loads off and put the ECU into Parking mode. This ensures that the system is in a safe, low-power state, and prevents unnecessary power consumption or potential faults.*

**Table 6. ASS module functions**

| ASS module function         | Description                                                  |
|-----------------------------|--------------------------------------------------------------|
| void ASS_sev_initTask(void) | It initializes the ASS task and related resources            |
| void ASS_sev_runTask(void)  | It triggers the ASS task to start the assembly procedure     |
| void ASS_sev_checker(void)  | It checks the state of the ASSE_mode pin by reading the GPIO |

### 4.2.13 PARK\_mng of the service layer

The Park module is responsible for managing the transition of the [AEK-POW-PDUMINI](#) ECU into a safe, low-power Parking state. This operation is essential before vehicle shutdown, during maintenance, or whenever the system must be de-energized in a controlled and safe manner. The key implemented functions are:

- Safe system standby: it initiates a coordinated standby procedure for all e-fuse devices and power management ICs, ensuring that all loads are safely disconnected and the system is ready for low-power operation.
- Inter-module coordination: it orchestrates the interaction between e-fuse (for channel/device standby) and PMIC (for VBATT standby), guaranteeing that all safety and operational constraints are respected.
- Wake-up policy management: it defines and manages the conditions under which the ECU can be safely reactivated from Parking mode.

All the interactions with the Park module are performed using asynchronous message queues. Other service layer modules (such as COMM or diagnostics) communicate with the Park module by placing structured request messages into a dedicated request queue. Each message specifies the operation to be performed (for example, enter Parking mode). The Park module includes a dedicated freeRTOS task (PARKTask) that continuously monitors the request queue, processes incoming requests, and coordinates the standby procedure.

A central aspect of the Park module is its interaction with the eFuse module, specifically through the EFUSE\_sev\_setStandby() function. This function is designed to safely transition all e-fuse devices and channels into standby mode. However, it does not force devices into standby unconditionally. Instead, it performs a series of safety checks to ensure that entering standby does not cause unsafe or damaging conditions. The checks performed depend upon the device:

- [L99SP08](#) devices: it checks if the current output on any channel exceeds the ITHx threshold (the programmable current limit). If any channel is above threshold, the device does not enter standby, as this indicates an active load or fault condition. In this case, the system does not enter standby, and the shutdown sequence is aborted to prevent unsafe conditions. Otherwise, all e-fuse devices are transitioned to Standby mode. The Park module then issues the VBATT standby command to the PMIC [SPSB081](#) PMIC (AEK\_POW\_SPSB081\_Force\_VBAT\_StandBy(SPSB081\_DEV0)). This command puts [SPSB081](#) into Standby mode, disabling both V1 and V2 outputs (as per datasheet [DS14048](#)). As a consequence of [SPSB081](#) entering standby, the microcontroller loses power and shuts down.

After entering parking mode, the system can be woken up in the following ways:

- CAN message to [SPSB081](#): the sending of a specific CAN message to [SPSB081](#) triggers its exit from standby and re-enables V1/V2 outputs, thus powering up the other PMIC and the microcontroller.
- Fault condition on e-fuse devices: if the current threshold is exceeded on an [L99SP08](#) channel, the corresponding DIAG pin is asserted, which can also be used as a wake up source.

Upon microcontroller wake-up, all e-fuse devices and PMICs automatically return to Normal mode. The system resumes full operation, ready to handle new commands or events. Typical interactions with the Park module include:

- e-fuse: the Park module's core logic is built around the `EFUSE_sev_setStandby()` function, which ensures all e-fuse devices are in a safe state before power-down.
- PMIC: it issues the VBATT standby command to [SPSB081](#) and it powers down the [SPSA068](#).
- COMM: it can receive parking requests via CAN and send status notifications.
- ERR: it might be used to log or handle errors if the parking procedure fails or is aborted due to unsafe conditions.

The following figure lists the functions implemented for the Park module.

**Figure 135. Park module functions**

| PARK module function                                            | Description                                              |
|-----------------------------------------------------------------|----------------------------------------------------------|
| <code>void PARK_sev_initTask(void);</code>                      | It initializes the PARK task and related resources.      |
| <code>void PARK_sev_runTask(void);</code>                       | It triggers the PARK task to start the parking procedure |
| <code>void PARK_sev_addReqQueue(PARK_sev_msgReq_t *req);</code> | It adds a parking request to the queue..                 |

The Park task logic (PARKTask) receives a parking request (from CAN, user command, or another module). It calls the `EFUSE_sev_setStandby()` function to safely transition all e-fuse devices to standby (only if all safety conditions are met). Finally, it issues the VBATT standby command to [SPSB081](#) PMIC. Once Parking mode is completed, it monitors wake-up events and manages the transition back to operational mode as required.

#### 4.2.14 ERR\_mng of the service layer

The error management (ERR) module is a critical component of the [STEVAL-PDUBV1](#) firmware, responsible for the detection, classification, storage, and reporting of all errors that occur within the ECU. It ensures that every relevant fault is tracked, its status is updated in real time, and that diagnostic information is available for UDS and after-sales analysis. The key implemented functions are:

- Error detection and classification: it monitors all relevant system events and conditions, classifies errors as ACTIVE or NOT\_ACTIVE, and updates their status accordingly.
- Diagnostic trouble code (DTC) management: it converts detected errors into DTCs (as per UDS standards) and manages their life cycle.
- EEPROM storage: it stores DTCs persistently in EEPROM, ensuring that error history is retained across power cycles.
- UDS integration: it provides DTC data and error status to the UDS diagnostic module for external querying and reporting.
- Interaction with other modules: it receives error notifications from modules such as eFuse, PMIC, and manages their status and reporting.

An error is considered ACTIVE when the condition that triggered it is still present (for example, a voltage is still out of range, or a device is still in fault). An error becomes NOT\_ACTIVE when the triggering condition is no longer present (for example, the voltage returns within range or the device recovers). The ERR module continuously updates the status of each error based on real-time system feedback. Each error is mapped to a diagnostic trouble code (DTC) according to UDS standards, allowing external diagnostic tools to query and interpret error conditions. All DTCs are stored in EEPROM, ensuring that error history is preserved across power cycles and available for after-sales diagnostics. The ERR module updates the ACTIVE/NOT\_ACTIVE status of each DTC in real-time, based on the current system state. Typical interactions with the ERR module include:

- eFuse and PMIC: the ERR task monitors and logs errors reported by these modules.
- UDS: it provides DTC data and error status for external diagnostics.
- COMM: it can send error notifications or status updates via CAN.

The following figure lists the functions implemented in the ERR module.

**Figure 136. ERR module functions**

| ERR module function                                                                                                 | Description                                                         |
|---------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| <code>void ERR_sev_initTask(void);</code>                                                                           | It initializes the ERR tasks, queues, and semaphores.               |
| <code>void ERR_sev_runTask(void);</code>                                                                            | It triggers the main ERR task (typically from an ISR or scheduler). |
| <code>void ERR_sev_setDTC(ERR_sev_errorCode_t code, ERR_sev_errorSts_t status, EFUSE_sev_channel_t channel);</code> | It sets or updates a DTC with the given code, status, and channel   |
| <code>void ERR_sev_clearDTC(void);</code>                                                                           | It clears all stored DTCs.                                          |
| <code>uint32_t ERR_sev_getActiveDTCNum(void);</code>                                                                | It returns the number of currently ACTIVE DTCs.                     |
| <code>uint32_t ERR_sev_getDTCNum(void);</code>                                                                      | It returns the total number of DTCs.                                |
| <code>ERR_sev_dtc_t ERR_sev_getDTC(void);</code>                                                                    | It retrieves a DTC.                                                 |
| <code>ERR_sev_dtc_t ERR_sev_getActiveDTC(void);</code>                                                              | It retrieves an ACTIVE DTC.                                         |

All DTCs are written to and read from EEPROM for persistent storage. The following table summarizes the function calls and their results.

**Table 7. Summary table for ERR interactions**

| Step | Module   | Action/function called                             | Result/effect                                  |
|------|----------|----------------------------------------------------|------------------------------------------------|
| 1    | ERR      | ERRVccTask wakes up, and calls setDTC()            | Error code set to ACTIVE, DTC updated          |
| 2    | ERR      | It writes the DTC into EEPROM                      | Error history is persistent                    |
| 3    | UDS/COMM | DTC is available for diagnostics and notifications | Error can be read by external tools or systems |

#### 4.2.15 List of diagnosable errors for ERR\_mng

Table 8 provides a detailed list of all errors that can be diagnosed and managed by the ERR module, as defined in the error code enumeration (ERR\_sev\_errorCode\_t). Each error is accompanied by a comment describing its meaning and typical cause.

**Table 8. ERR diagnosable errors**

| Error code enum      | Device  | Description                               |
|----------------------|---------|-------------------------------------------|
| PMIC_VSUV_SPSB081    | SPSB081 | Supply undervoltage detected              |
| PMIC_VSOV_SPSB081    | SPSB081 | Supply overvoltage detected               |
| PMIC_V1UV_SPSB081    | SPSB081 | V1 undervoltage detected                  |
| PMIC_V1OV_SPSB081    | SPSB081 | V1 overvoltage detected                   |
| PMIC_WDGFALL_SPSB081 | SPSB081 | Watchdog failure                          |
| PMIC_TW_SPSB081      | SPSB081 | Thermal warning                           |
| EFUSE_OVC            | e-fuse  | Overcurrent detected                      |
| EFUSE_OVT            | e-fuse  | OVT detected                              |
| EFUSE_WDGFALL        | e-fuse  | Watchdog failure                          |
| EFUSE_TW             | e-fuse  | Thermal warning                           |
| EFUSE_VDSHSR         | e-fuse  | High-side short detected (VDSHSR)         |
| EFUSE_ITLOFFSR       | e-fuse  | ITL off detected                          |
| EFUSE_CHLOFFSR       | e-fuse  | CHL off detected                          |
| EFUSE_ITST           | e-fuse  | ITST detected                             |
| EFUSE_OTSR           | e-fuse  | Overtemperature detected                  |
| EFUSE_IPEAKLSR       | e-fuse  | I <sub>PEAK</sub> low-side short detected |

| Error code enum  | Device  | Description                           |
|------------------|---------|---------------------------------------|
| EFUSE_LHIRB      | e-fuse  | Retrieves limp Home IRB status        |
| PCHANNEL_WDFAIL  | L99SP08 | Watchdog failure                      |
| PCHANNEL_BGFAIL  | L99SP08 | Bandgap reference failure             |
| PCHANNEL_OSCFAIL | L99SP08 | Oscillator failure                    |
| PCHANNEL_VCCOV   | L99SP08 | VCC overvoltage                       |
| PCHANNEL_VCCUV   | L99SP08 | VCC undervoltage                      |
| PCHANNEL_TW      | L99SP08 | Thermal warning                       |
| PCHANNEL_VDSMAX  | L99SP08 | Maximum drain-source voltage exceeded |

#### 4.2.16 UDS\_mng of the service layer

The unified diagnostic services (UDS) module implements a simplified version of the ISO14229 standard, providing diagnostic and service access to the [AEK-POW-PDUMINI](#) ECU via CAN. It enables external tools and testers to interact with the ECU for configuration, fault reading, and system status, using a standardized protocol widely adopted in the automotive industry. The key implemented functions are:

- Session management: it supports different diagnostic sessions (for example, default, extended) to control access to various services and data.
- Service identifier (SID) handling: it decodes and processes UDS service requests (SIDs) such as session control, DTC reading, DTC clearing, and data reading.
- Data identifier (DID) handling: it provides access to specific data items (DIDs) such as channel status, current, voltage, temperature, and error information.
- Positive and negative response generation: it returns standardized positive or negative responses to each UDS request, as per ISO14229.
- Integration with other modules: it interacts with ERR, eFuse, and other modules to retrieve diagnostic data and execute service requests.

Typical interactions with the UDS module include:

- ERR: it provides DTC data and error status for UDS DTC services.
- e-fuse: it provides channel status, current, voltage, temperature, and other data for UDS data services.
- COMM: it receives UDS requests via CAN and sends UDS responses.

The following figure lists the functions implemented for the UDS module.

**Figure 137. UDS module functions**

| UDS module function                                           | Description                                            |
|---------------------------------------------------------------|--------------------------------------------------------|
| <code>void UDS_sev_initTask(void);</code>                     | It initializes the UDS task and related resources.     |
| <code>void UDS_sev_runTask(void);</code>                      | It triggers the UDS task to process incoming requests. |
| <code>void UDS_sev_addReqQueue(UDS_sev_msgReq_t *req);</code> | It adds a UDS request to the queue.                    |

The task logic (UDSTask) receives UDS requests from the communication module, decodes the SID and DID, and interacts with ERR, eFuse, and other modules to retrieve or modify data. The task sends positive or negative responses according to ISO14229.



#### 4.2.17 UDS sessions, SIDs, and DIDs

The UDS sessions define the diagnostic access level and available services. The default session (DEFAULT) allows basic diagnostic access with limited services. The extended diagnostic session (EXTENDED\_DIAGNOSTIC) grants full access to advanced diagnostics, DTC management, and data reading. SIDs are 1-byte codes that identify the requested UDS service. The UDS module implements the following SIDs.

**Table 9. SID codes**

| SID (Hex) | Service name            | Description                                |
|-----------|-------------------------|--------------------------------------------|
| 0x10      | Session control         | Switches between diagnostic sessions       |
| 0x14      | Clear DTC               | Clears all stored diagnostic trouble codes |
| 0x19      | Read DTC                | Reads DTCs and their status                |
| 0x22      | Read data by identifier | Reads specific data items (DIDs)           |

DIDs are 2-byte codes that specify the data item to be accessed. The UDS module implements the following DIDs.

**Table 10. Examples of DID codes**

| DID (Hex) | Data item             | Description                                                    |
|-----------|-----------------------|----------------------------------------------------------------|
| 0xF110    | Channel status        | It returns the status (on/off) of a channel                    |
| 0xF111    | Channel current       | It returns the current of a channel                            |
| 0xF112    | Channel voltage       | It returns the voltage of a channel                            |
| 0xF113    | Channel temperature   | It returns the temperature of a channel                        |
| 0xF114    | Channel ITLOff status | It returns I <sup>2</sup> t event feedback status of a channel |
| 0xF115    | Channel DIEN status   | It returns DI input pin status                                 |
| 0xF116    | Channel OVC           | It returns overcurrent status of a device                      |
| 0xF117    | Channel OVT           | It returns overtemperature status of a device                  |
| 0xF119    | Channel VDSMax        | It returns voltage drain source max flag for a channel         |
| 0xF121    | Channel CHLOFF        | It returns a flag in case of power limitation of overcurrent   |
| 0xF122    | Channel IPEAK         | It returns a flag in case of Ipeak detected                    |
| 0xF123    | Channel WDGFail       | It returns a flag in case of watchdog failure                  |
| 0xF124    | P-Channel WDGFail     | It returns a flag in case of P-Channel watchdog failure        |
| 0xF126    | Channel LCC           | It returns the LCC status of a channel                         |
| 0xF128    | SPSB081 WDGFail       | It returns a flag in case of SPSB081 watchdog failure          |

All the implemented SIDs and DIDs in the **AEK-POW-PDUMINI** stack software are listed below:

- Session control (SID 0x10): it switches between default and extended diagnostic sessions.
  - Example request: SID: 0x10, Data: 0x03 (EXTENDED\_DIAGNOSTIC).
  - Positive response: SID: 0x50 (0x10 + 0x40) echo session type, timing parameters.
  - Negative response: SID: 0x7F, Data: 0x10 (original SID), error code (for example, 0x12 for subfunction not supported).
- Clear DTC (SID 0x14): it clears all DTCs stored in the ERR module and EEPROM.
  - Example request: SID: 0x14.
  - Positive response: SID: 0x54 (0x14 + 0x40) confirmation of DTC clear.
  - Negative response: SID: 0x7F, Data: 0x14, error code (for example, 0x22 for wrong conditions).
- Read DTC (SID 0x19): it reads the list of DTCs and their status (ACTIVE/NOT\_ACTIVE).
  - Example request: SID: 0x19, Data: 0x01 (report number of DTCs), 0xFF (all statuses).
  - Positive response: SID: 0x59 (0x19 + 0x40) number of DTCs, DTC codes, status.
  - Negative response: SID: 0x7F, Data: 0x19, error code (for example, 0x31 for request out of range).
- Read data by identifier (SID 0x22): it reads specific data items (DIDs) such as channel status, current, voltage, temperature, LCC, etc.
  - Example request: SID: 0x22, Data: 0xF1 0x10 (DID for channel status), channel number.
  - Positive response: SID: 0x62 (0x22 + 0x40) DID, channel, value.
  - Negative response: SID: 0x7F, Data: 0x22, error code (for example, 0x31 for request out of range, 0x22 for wrong conditions).

**Table 11. UDS message examples**

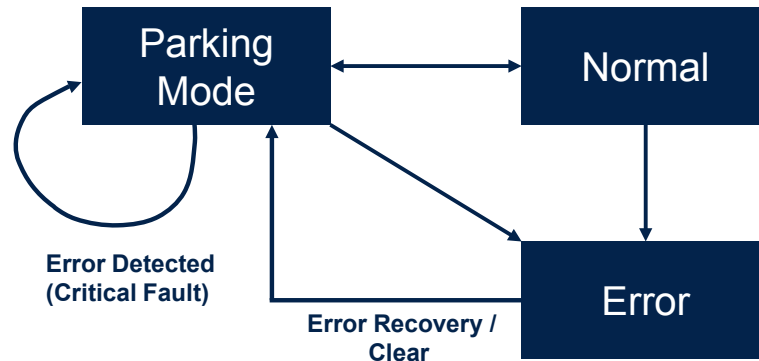
| Message                               | Request     | Positive response                                                                                                                                         | Negative response                                                                          |
|---------------------------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| Switch to extended diagnostic session | 10 03       | 50 03 00 32 01 F4 00 00<br>50: Positive response SID<br>03: Session type (extended)<br>0032: P2 server timing<br>01F4: P2* extended timing                | 7F 10 12<br>7F: Negative response SID<br>10: Original SID<br>12: Subfunction not supported |
| Clear all DTCs                        | 14          | 54 00 00 00 00 00 00 00<br>54: Positive response SID<br>Remaining bytes: confirmation                                                                     | 7F 14 22<br>7F: Negative response SID<br>14: Original SID<br>22: Wrong conditions          |
| Report number of DTCs in all statuses | 19 01 FF    | 59 01 00 00 00 02 00 00<br>59: Positive response SID<br>01: Subfunction<br>00000002: Number of DTCs                                                       | 7F 19 31<br>7F: Negative response SID<br>19: Original SID<br>31: Request out of range      |
| Read status of channel 6              | 22 F1 10 06 | 62 F1 10 06 01 00 00 00<br>62: Positive response SID<br>F1 10: DID for channel status<br>06: Channel number<br>01: Status (enabled)                       | 7F 22 31<br>7F: Negative response SID<br>22: Original SID<br>31: Request out of range      |
| Read current of channel 6             | 22 F1 11 06 | 62 F1 11 06 00 64 00 00<br>62: Positive response SID<br>F1 11: DID for channel current<br>06: Channel number<br>0064: Current value (for example, 100 mA) | 7F 22 31<br>7F: Negative response SID<br>22: Original SID<br>31: Request out of range      |

**Note:** For further details, see [Section Appendix A: UDS extension and threshold configuration](#).

## 5 AEK-POW-PDUMINI software state machine

The figure below shows the state machine for the AEK-POW-PDUMINI software.

Figure 138. Application software state machine



### 5.1 Normal mode

Normal mode is the standard operational state of the AEK-POW-PDUMINI, where all modules (eFuse, PMIC, COMM, UDS, etc.) are fully functional and manage loads and channels according to the system logic. In this mode, power distribution, diagnostics, and protection mechanisms are fully active. The system enters Normal mode after a wake-up event, when:

- An exceeding current threshold is present in one of the L99SP08 channels, typically when the load demand surpasses the standby capability.
- A CAN message requesting system activation is received (for example, a wake-up command from the vehicle network).
- Wake-up from Parking mode is requested via a hardware fault event.
- There is a recovery from an ERROR state, that is, all critical faults have been cleared, and all modules have returned to a safe operational state.

While in Normal mode, the following policies are active:

- Power channel management: both the P-channel switches (L99SP08) and the e-fuses operate in Normal mode. With the I<sup>2</sup>t protection, each channel is monitored according to its programmed energy curve, preventing overheating of wiring and components. In addition, according to thermal policy, junction and case temperatures are continuously monitored and, if thresholds are exceeded, shutdown or derating policies are applied according to the configuration. Each channel must respect its nominal current, and any overcurrent is managed according to the configured protection policy (latch-off, autorestart, etc.).
- Load management: all loads are powered and managed according to system logic and application requirements. Diagnostics are active; therefore, any anomalies are reported via the UDS protocol or through status registers. When a Parking mode request is received, the system evaluates load priorities: only non-essential loads are powered down, while critical loads remain supplied until it is safe to disconnect them.
- State transition handling: the system continuously monitors the status of all channels, temperatures, currents, and voltages, and reacts promptly in case of anomalies. All state transitions are subject to safety checks to prevent unintended power loss to sensitive or safety-critical loads.
- Diagnostics and reporting: all protection events (I<sup>2</sup>t, thermal, overcurrent) are logged and can be reported via UDS or diagnostic interfaces for traceability and maintenance.

While in Normal mode, the system can transition to:

- Parking mode: transition to Parking mode is only allowed if all e-fuses are no longer required (that is, no critical load is active), and the total load current has dropped below the threshold manageable by L99SP08 in Standby mode (typically below I<sub>THR</sub>). Any parking/shutdown request via CAN, user command, or system logic is considered only if these safety and operational conditions are met, ensuring that essential loads remain powered as long as necessary.
- ERROR: if a critical fault is detected (for example, persistent out-of-range voltage, unrecoverable e-fuse or PMIC error), the system transitions to Error mode, disabling loads according to the safety policies.

## 5.2 Parking mode

In Parking mode, AEK-POW-PDUMINI is in a safe, low-power state. All e-fuse channels (for example, VNF9x family) are powered down as the microcontroller is off, and the PMICs are in Standby mode. The P-channel switch (L99SP08) is in Standby mode too. In this state, all channels connected with essential always-on loads remain powered as they were in Normal mode (for example, telematics, security, or gateway modules). Only wake-up channels are monitored to allow the system to resume operation when required.

To enter Parking mode, a Park request shall be received and all safety checks in setStandby shall pass (for example, no overcurrent, all e-fuses can be safely powered down).

The specific actions connected to Parking mode are:

- e-fuse channels (VNF9x): all e-fuse channels are powered down, as the microcontroller is not operational and cannot manage them.
- PMIC: SPSB081 is in Standby mode, supplying only the minimal power required for the wake-up logic.
- P-channel switch (L99SP08): the P-channel switch enters Standby mode. In this state, any channels that were active during Normal mode remain powered, ensuring continuity for always-on loads.
- Microcontroller: the microcontroller is powered down and does not execute the code.
- Wake-up monitoring: only wake-up sources (CAN or hardware fault events) are monitored.

While in Parking mode, the system can transition to:

- Normal: upon a valid wake-up event, such as a CAN message addressed to the PMIC or a hardware fault event (for example, overcurrent on a P-channel detected via DIAG pin).
- Error: if a fault is detected during the transition to Parking mode, and the system cannot safely enter standby (for example, a channel cannot be safely powered down, or a persistent fault is present). The system has detected a critical fault (for example, voltage out-of-range, e-fuse/PMIC unrecoverable error, persistent hardware fault). Error becomes ACTIVE and DTC is set. The same happens when any module (typically, eFuse, PMIC) detects a critical error and notifies ERR, which sets the error to ACTIVE.
- Parking: when a failed attempt to enter Parking mode due to unsafe conditions is recorded. This could go back to Normal if all errors are cleared (for example, voltage returns in range, faults are resolved, and DTCs are cleared).

If a programmable current threshold is exceeded on any P-channel (L99SP08) during Parking mode, we get a corresponding DIAG pin transition (typically towards LOW). This DIAG signal is routed to the PMICs, which interpret it as a wake-up event. The PMIC then powers up the microcontroller, which resumes operation and transitions the system from Parking mode to Normal. This mechanism ensures that if an unexpected load or fault condition occurs on an always-on channel, the system can immediately wake up and take appropriate action.

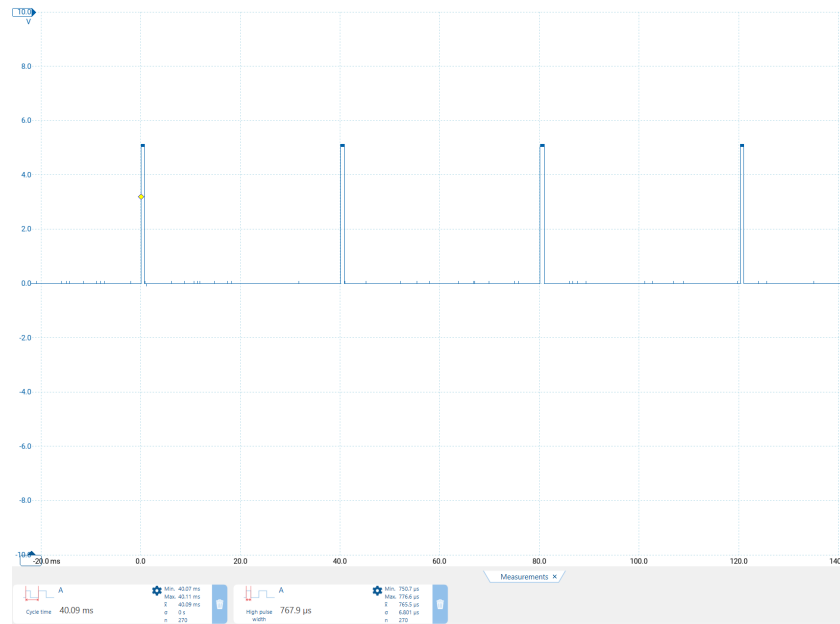


### 5.3 Task scheduling and duration

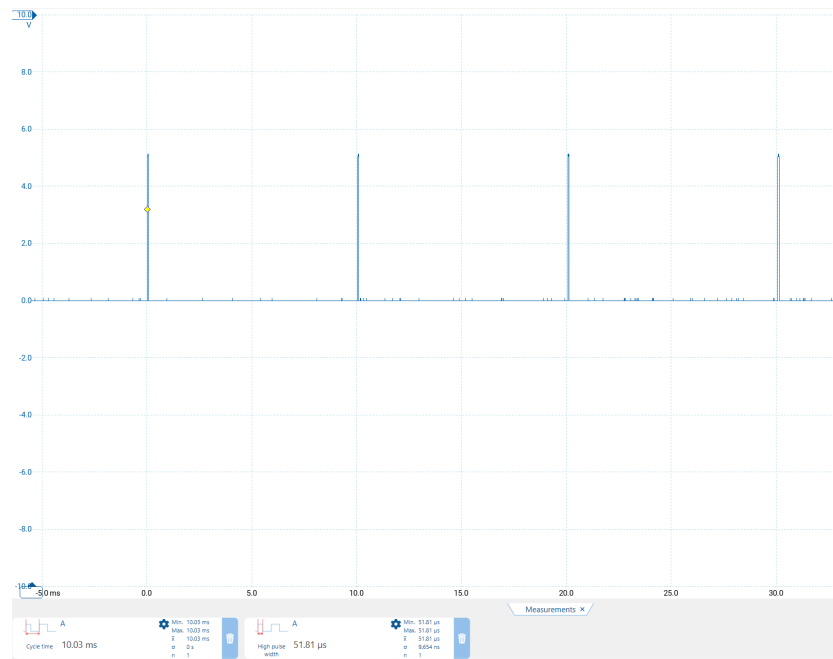
The figures below show the typical scheduling for the service layer main tasks:

- eFuse task: task scheduled every 40.9  $\mu$ s with a duration of 767.9  $\mu$ s.

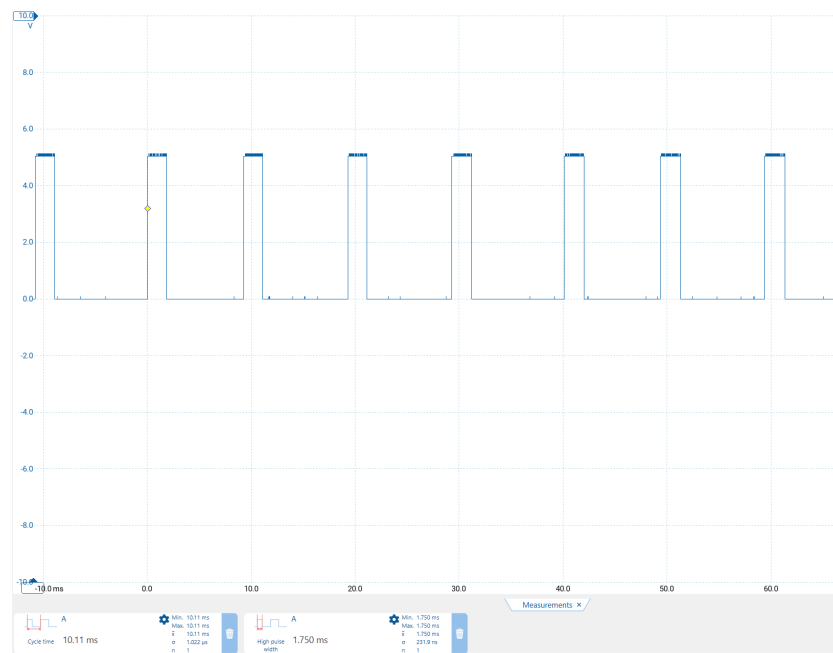
**Figure 139. eFuse task timing diagram**



- PMIC task: task scheduled every 10.03 msec with a duration of 51.81  $\mu$ s.

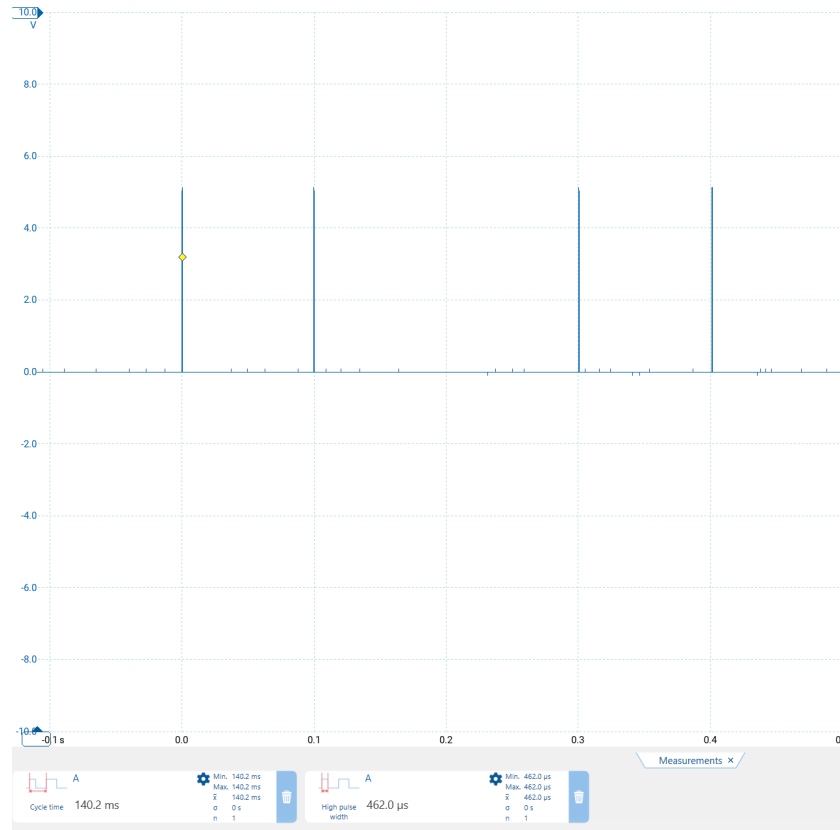
**Figure 140. PMIC task timing diagram**


- ERR task: task scheduled every 10.11 msec with a duration of 1.750 msec.

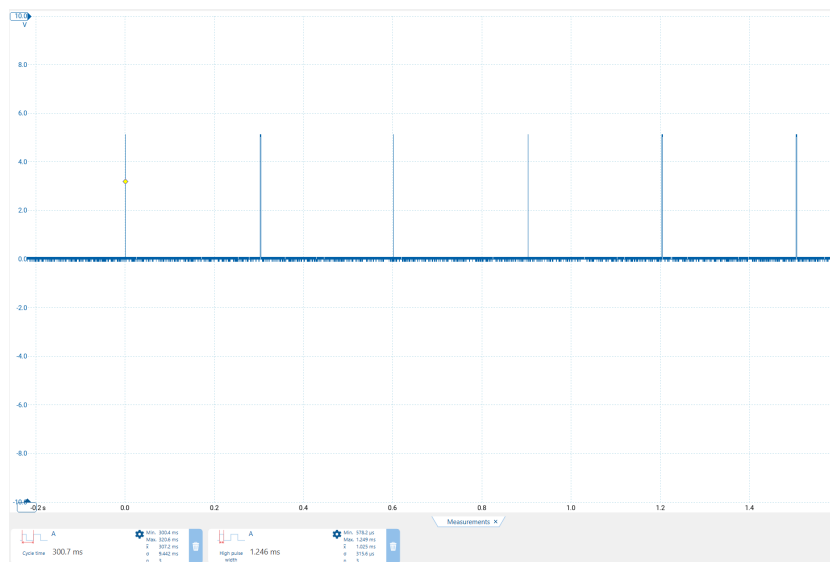
**Figure 141. ERR task timing diagram**


- COMMRx task: task scheduled every 140.2 msec (CAN message sent from host every 140 msec) with a duration of 462.0  $\mu$ sec.

**Figure 142. COMMRx task timing diagram**



- Park task: task scheduled every 300.7 msec (CAN sent from host every 300 msec) with a duration of 1.246 msec.

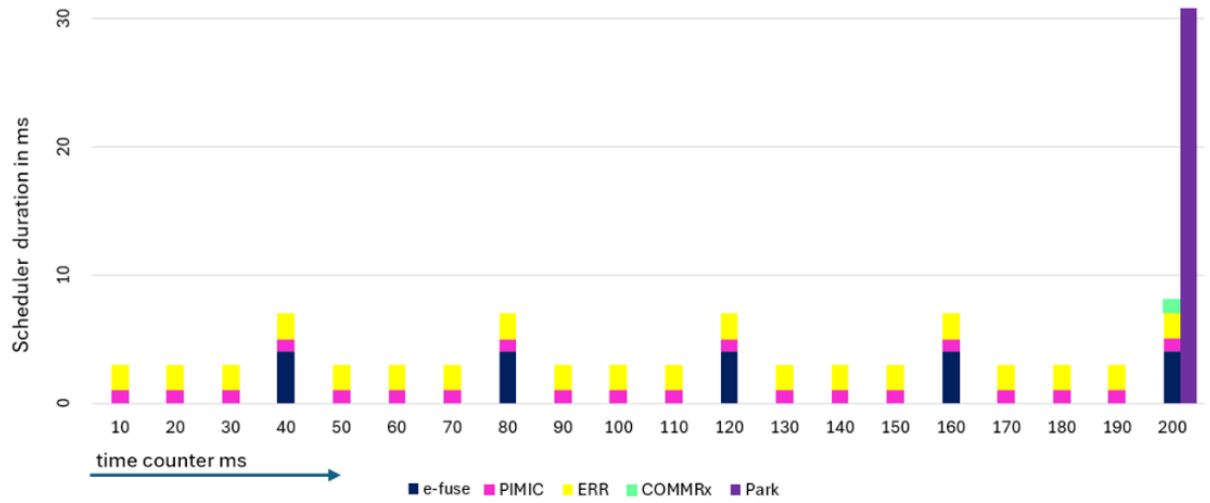
**Figure 143. Park task timing diagram**

**Figure 144. Sti<sup>2</sup>Fuse PDUB software timing**

| Time | EFUSE | PMIC | ADCWDT | ERR | CommRx | Park |
|------|-------|------|--------|-----|--------|------|
| 10   |       | 1    |        | 2   |        |      |
| 20   |       | 1    |        | 2   |        |      |
| 30   |       | 1    |        | 2   |        |      |
| 40   | 4     | 1    |        | 2   |        |      |
| 50   |       | 1    |        | 2   |        |      |
| 60   |       | 1    |        | 2   |        |      |
| 70   |       | 1    |        | 2   |        |      |
| 80   | 4     | 1    |        | 2   |        |      |
| 90   |       | 1    |        | 2   |        |      |
| 100  |       | 1    | 1      | 2   | 1      |      |
| 110  |       | 1    |        | 2   |        |      |
| 120  | 4     | 1    |        | 2   |        |      |
| 130  |       | 1    |        | 2   |        |      |
| 140  |       | 1    |        | 2   |        |      |
| 150  |       | 1    |        | 2   |        |      |
| 160  | 4     | 1    |        | 2   |        |      |
| 170  |       | 1    |        | 2   |        |      |
| 180  |       | 1    |        | 2   |        |      |
| 190  |       | 1    |        | 2   |        |      |
| 200  | 4     | 1    | 1      | 2   | 1      | 31   |
| 210  |       | 1    |        | 2   |        |      |
| 220  |       | 1    |        | 2   |        |      |



According to the values in Figure 144, the complete time diagram shall be like:

Figure 145. All task timing diagram



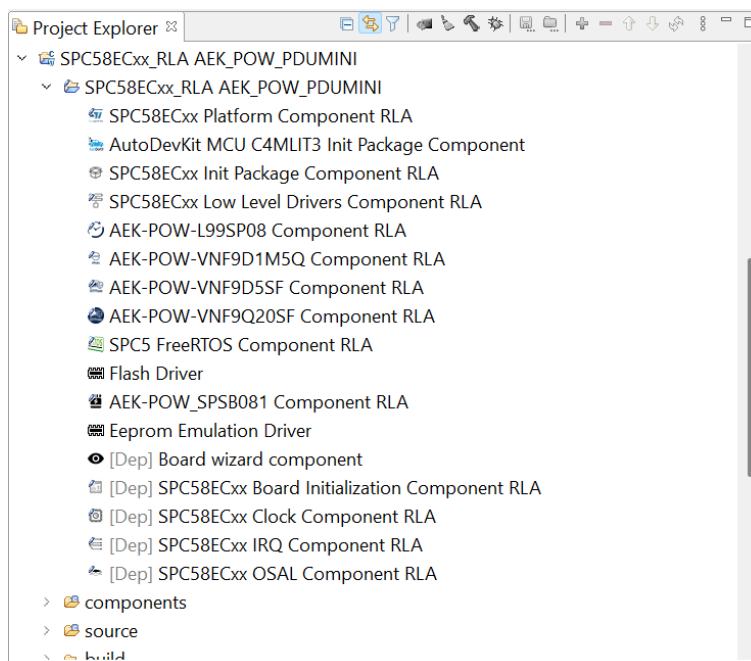
## 6 Demos available in AutoDevKit

### 6.1 SPC58ECxx\_RLA AEK\_POW\_PDUMINI demo

Starting from release 2.7.0, the SPC58ECxx\_RLA AEK\_POW\_PDUMINI demo for the AEK-POW-PDUMINI evaluation board is available in AutoDevKit.

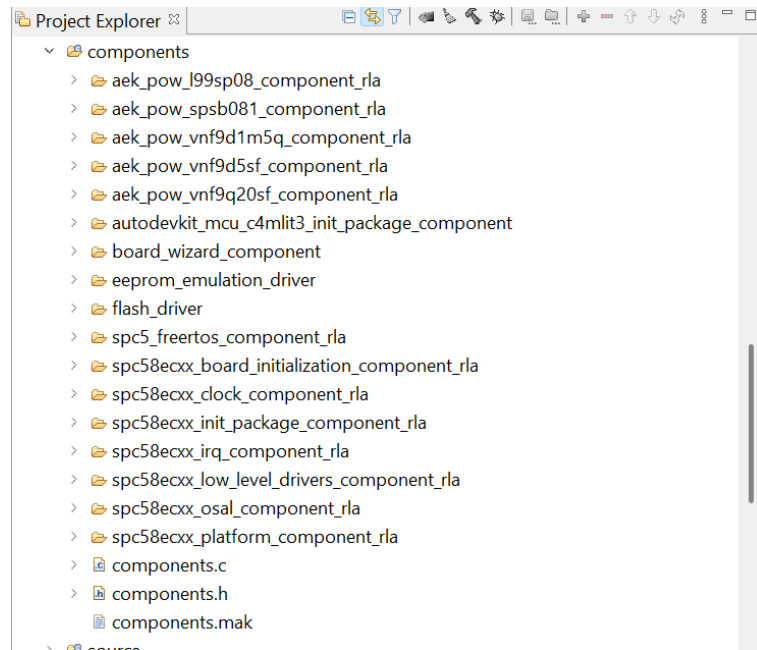
The demo is included in the AEK\_POW\_L99SP08 component and it consists of several components, as shown in the figure below.

**Figure 146. SPC58ECxx\_RLA AEK\_POW\_PDUMINI in AutoDevKit**



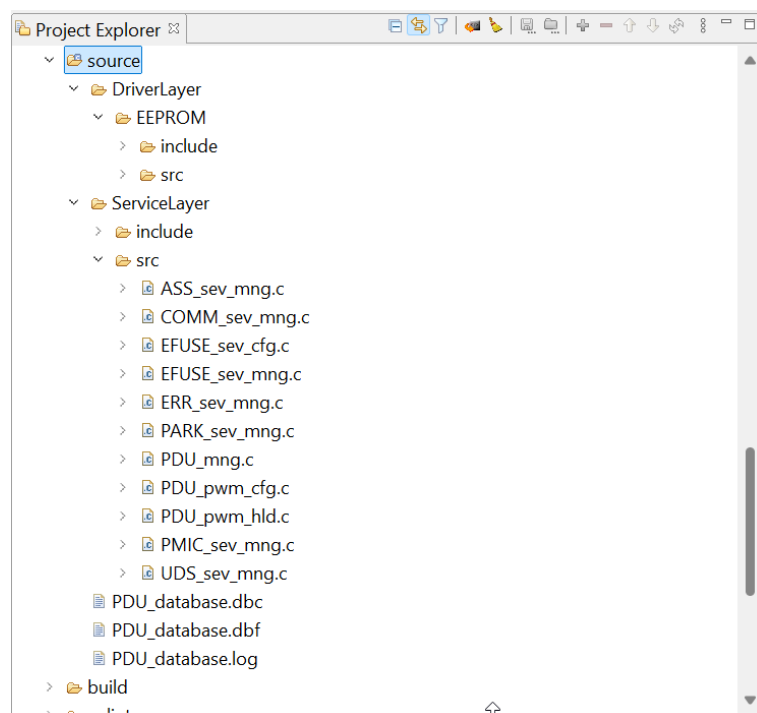
The components folder includes all drivers and configuration files (.cfg), which are automatically generated by AutoDevKit.

Figure 147. Demo components folder



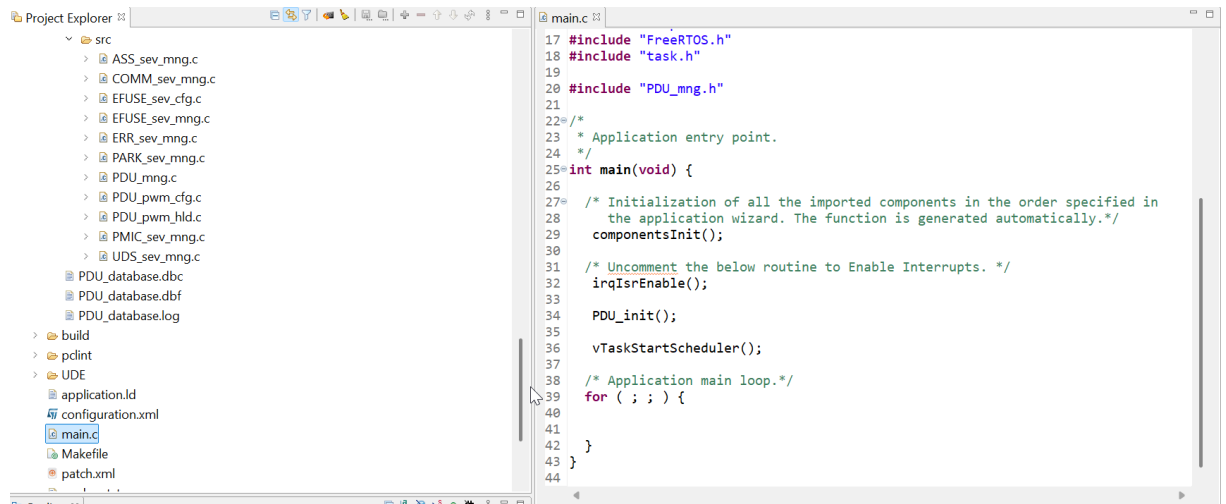
The source folder includes the service layer, driver layer, and dbCan file (PDU\_database.dbc).

Figure 148. Source folder



The main.c file initializes the components and the board. It also performs task initialization.

Figure 149. main.c file



The SPC58ECxx\_RLA AEK\_POW\_PDUMINI demo is based on freeRTOS and implements a service layer to manage a power distribution unit (PDU) in an automotive environment.

The system is split into independent modules that communicate through tasks, queues, and semaphores, handling diagnostics, communication, power control, error management, and safety procedures.

The main components are:

- AEK-POW-L99SP08: driver for P-channel.
- AEK-POW-VNF9D1M5Q, AEK-POW-VNF9D5SF, AEK-POW-VNF9Q20SF: drivers for e-fuse power devices and their channels.
- SPC5 freeRTOS: real-time operating system for task management.
- Flash driver: management of flash memory for non-volatile data.
- AEK-POW-SPSB081: driver for the [SPSB081](#) power management IC (PMIC).
- EEPROM emulation driver: EEPROM emulation on flash for diagnostic trouble codes (DTCs) storage.

The project is structured into several modules. Each module is responsible for a specific functionality, as described in the following sections.

### 6.1.1 UDS service manager

The UDS service manager (UDS\_sev\_mng.c) manages unified diagnostic services (UDS) requests.

It handles diagnostic sessions, both in default and extended modes, and can provide positive or negative responses in accordance with the UDS protocol.

This module interfaces directly with the eFuse and PMIC modules to collect diagnostic data, using freeRTOS queues and semaphores to ensure robust and thread-safe communication.

### 6.1.2 PMIC service manager

The PMIC service manager (PMIC\_sev\_mng.c) manages all interactions with the power management IC ([SPSB081](#)).

It is structured into two distinct tasks: one for general management and one for control operations.

This module supports asynchronous handling of requests and responses through dedicated queues and is responsible for the PMIC initialization and configuration.

Additionally, it provides seamless integration with other system modules via intertask communication.

### 6.1.3 e-fuse service manager

The e-fuse service manager (EFUSE\_sev\_mng.c) manages e-fuse channels, which are essential for power distribution and protection within the system.

It includes both management and control tasks and is responsible for initializing and configuring multiple e-fuse devices.

The module supports reading and writing various parameters, including current, voltage, temperature, and fault status, across multiple devices and channels.

Additionally, it implements standby and fail-safe functionalities to enhance system reliability.

#### 6.1.4 **Park service manager**

The Park service manager (PARK\_sev\_mng.c) manages the system safe parking sequence.

It coordinates the transition of the eFuse and PMIC modules into Standby mode via a dedicated task.

The sequence is executed within a critical section to prevent rush conditions, and the module can restore the error management subsystem if a failure occurs during the parking process.

#### 6.1.5 **ERR service manager**

The ERR service manager (ERR\_sev\_mng.c) manages error diagnostics and storage of diagnostic trouble codes (DTC).

This module operates through a dedicated task that periodically scans for errors in the PMIC and e-fuse subsystems.

It maintains DTCs both in RAM and EEPROM, providing functions for reading, clearing, and counting active DTCs.

The ERR service manager also supports stopping and resuming tasks, allowing flexible error management strategies.

#### 6.1.6 **COMM service manager**

The COMM service manager (COMM\_sev\_mng.c) facilitates communication among modules, managing message exchange via the CAN bus.

This module operates with separate tasks for message transmission and reception, using asynchronous message queues.

It is responsible for parsing and generating CAN frames and is tightly integrated with the eFuse, UDS, and Park modules to ensure consistent system operation.

#### 6.1.7 **ASS service manager**

The ASS service manager (ASS\_sev\_mng.c) verifies the system Assembly status.

It performs hardware verification through a dedicated status pin and manages the activation and deactivation sequence of the e-fuse channels.

Assembly check results are reported both via CAN messages and LED indicators, providing clear feedback on system status.

#### 6.1.8 **Operational flow**

At system startup, each module initializes its queues, semaphores, and tasks, establishing the infrastructure necessary for concurrent operation.

Communication between modules occurs through structured message exchanges via queues and notifications, ensuring thread-safe and deterministic data transfer.

During operation, various events, such as diagnostic requests, state transitions, or error conditions are handled asynchronously by the relevant modules, leveraging freeRTOS real-time capabilities.

Critical sequences, such as those involved in safe parking or error management, are protected using strict state checks, preventing rush conditions and ensuring system integrity.

Throughout operation, all errors are systematically tracked by the ERR Service manager and can be consulted or cleared via UDS requests, providing comprehensive diagnostic coverage and facilitating maintenance and troubleshooting.

### 6.2 **Additional demos**

1. Multi-Unit Power Distribution via CAN Bus
2. Remote Multi-Gateway Power Distribution Management over Ethernet



### 6.2.1 Multi-Unit Power Distribution via CAN Bus

This application manages four power distribution boards connected via a CAN bus: three [AEK-POW-PDUMINI](#) units and one [STEVAL-PDUBV1](#) unit. Each board is uniquely identified by an ID included in every CAN message transmitted over the UDS bus. Messages specify the target board's ID along with the service or channel to activate, enabling precise control and communication. This design ensures efficient, scalable, and reliable power distribution management across the entire system.

Each board is assigned a unique numeric identifier (PDU\_ECU\_ID). This identifier acts as an offset that is added to a set of base CAN message identifiers defined at system level.

All boards use the same set of logical messages, such as channel control, parking mode, assembly information, and diagnostics. However, each board's actual CAN identifiers are calculated by adding its PDU\_ECU\_ID to each base identifier.

This approach ensures that all boards use the same communication model and message set. Moreover, each board uses a distinct set of CAN identifiers, derived from the base identifiers shifted by the unique offset.

To configure a board, set its PDU\_ECU\_ID value in the CAN\_sev\_db.h file. Changing this numeric identifier updates the entire set of CAN message identifiers for that board. All other nodes can use the same firmware and structure, differing only by this per-node identifier.

On the same CAN bus, a client node sends messages using this addressing scheme. For each message, the client selects the base identifier for the required function and the target PDU\_ECU\_ID of the intended board. The final CAN identifier in the message is the sum of the base identifier and the target PDU\_ECU\_ID.

Each board processes only CAN frames with identifiers in its own shifted identifier set.

*Note: To use this new feature, download and run the following demo applications:*

1. `SPC58ECxx_RLA AEK_POW_PDUMINI_CAN_ID1`
2. `SPC58ECxx_RLA AEK_POW_PDUMINI_CAN_ID2`
3. `SPC58ECxx_RLA AEK_POW_PDUMINI_CAN_ID3`

### 6.2.2 Remote Multi-Gateway Power Distribution Management over Ethernet

This demo provides a simple, scalable example of how to convert CAN-only miniPDUs into Ethernet-reachable gateways. It preserves full UDS compatibility while adding remote, multi-gateway control and diagnostics with minimal incremental firmware complexity.

The demo manages three [AEK-POW-PDUMINI](#) power distribution units. Each unit is connected to an AEK-MCU-C4M2 (MCU board included in the soon available [AEK-MCU-C4MLIT2](#) evaluation kit) gateway acting as a TCP/IP server, plus a fourth AEK-MCU-C4M2-based Ethernet client.

Each board is assigned a unique IP address, MAC address, and PLCA ID.

The four AEK-MCU-C4M2 boards share a 10BASE-T1S PLCA segment: three boards act as miniPDU servers, and one board acts as an Ethernet client.

Each server exposes its miniPDU services (channel control and UDS diagnostics) on both CAN and a TCP/IP socket.

The client maintains three concurrent TCP connections and uses standard UDS services to monitor and toggle the PDU channels on all servers.

Ethernet is integrated directly into the existing communication manager (COMM\_sev\_mng.c). New RX and TX helpers translate TCP payloads into the same internal message format used for CAN. The FreeRTOS RX and TX tasks route messages based on a "transport type" field.

The UDS layer and miniPDU control logic remain unchanged and transport-independent, so the same UDS requests and responses work identically over CAN and Ethernet.

Based on the diagnostic responses, the client sequentially switches the channels ON and OFF across the three servers, enabling centralized, automated control and verification of the distributed PDUs.

*Note: To use this new application, download and run the following demos:*

*Server side (AEK-MCU-C4M2 plus attached AEK-POW-PDUMINI):*

1. `SPC58ECxx_RLA AEK_POW_PDUMINI_ETH0`
2. `SPC58ECxx_RLA AEK_POW_PDUMINI_ETH1`
3. `SPC58ECxx_RLA AEK_POW_PDUMINI_ETH2`

*Client side to be used with the AEK-MCU-C4M2:*

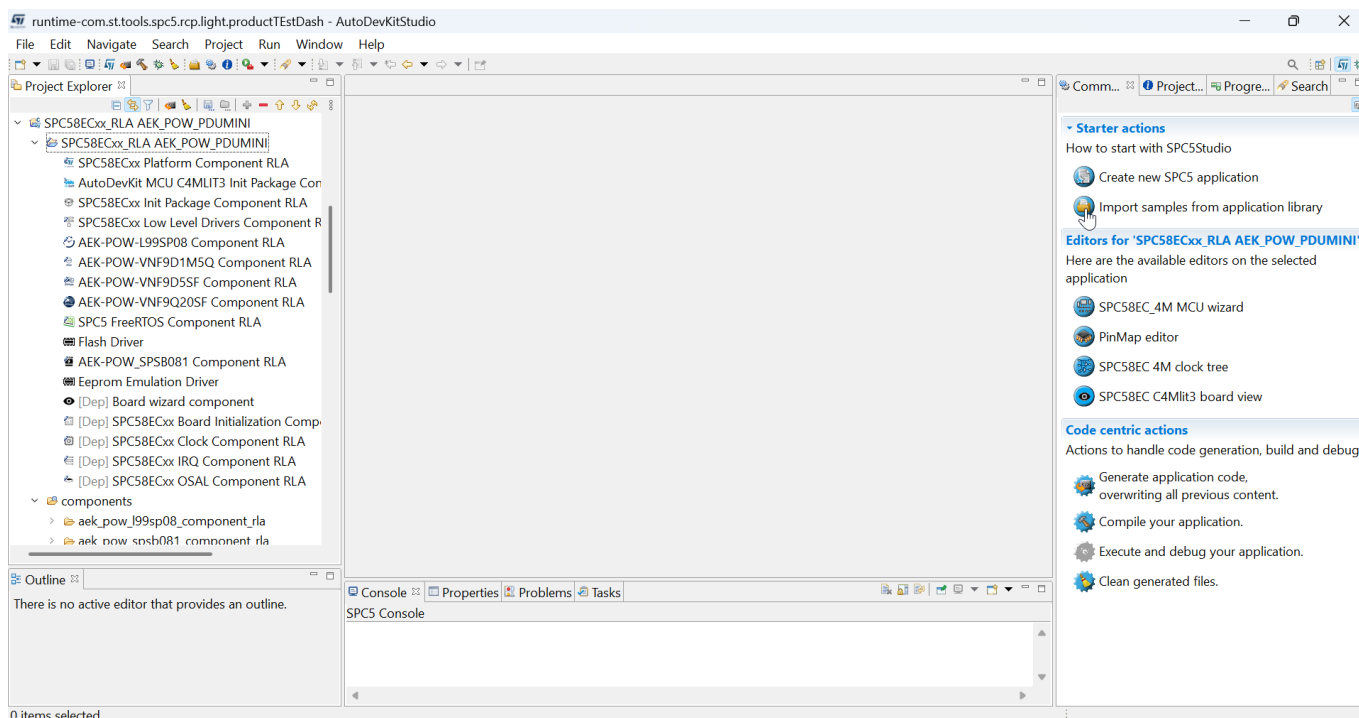
1. `AEK-MCU-C4MLIT2 Remote multi-Client miniPDU Ethernet Client`

## 6.3 How to import a demo from AutoDevKit

To import a demo from AutoDevKit, follow the steps below:

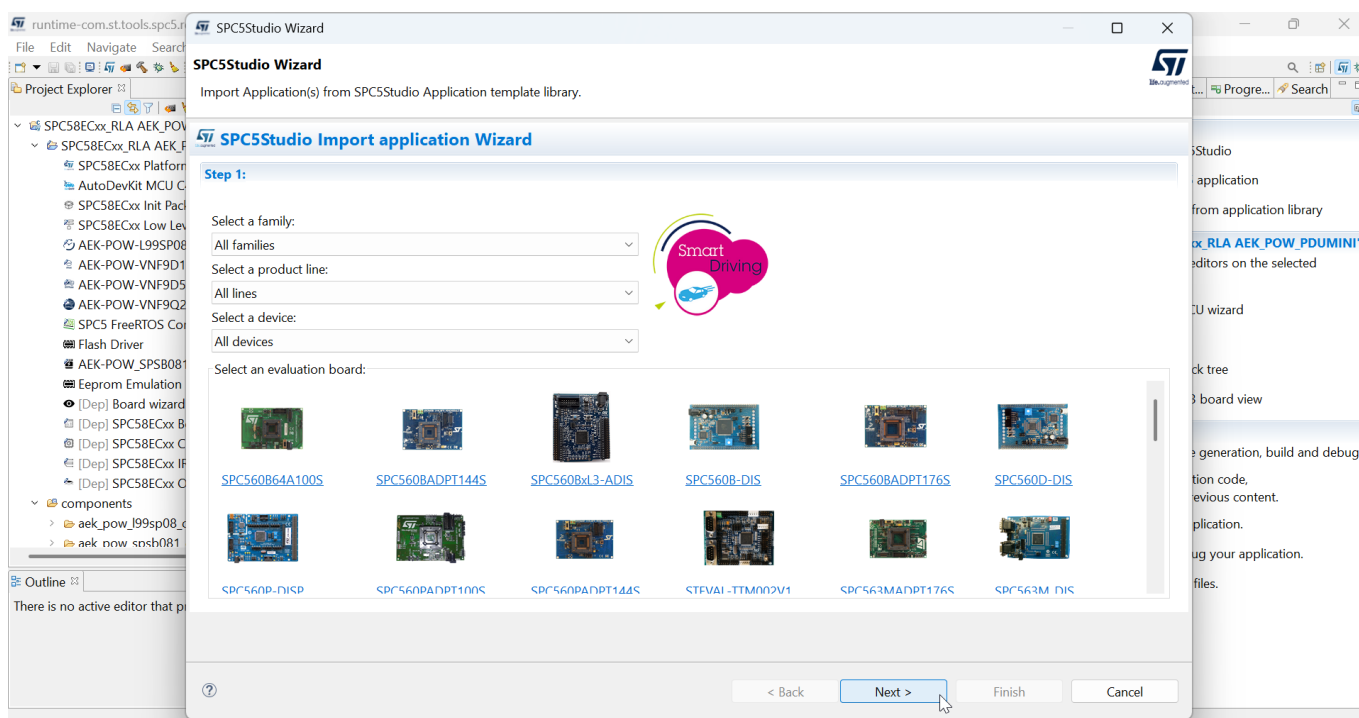
**Step 1.** Click on “Import samples from application library”.

**Figure 150. Importing the demo from the application library (1 of 4)**



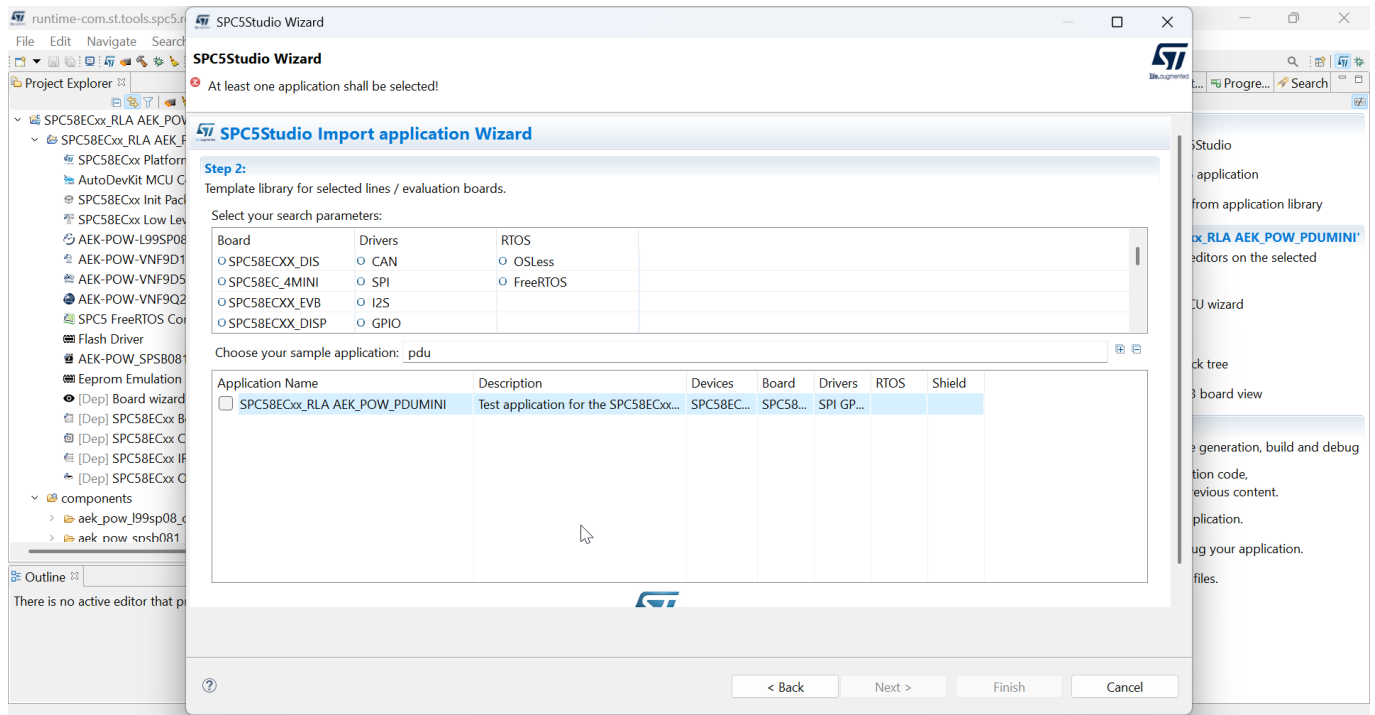
**Step 2.** In the pop-up window, click on “Next”.

**Figure 151. Importing the demo from the application library (2 of 4)**



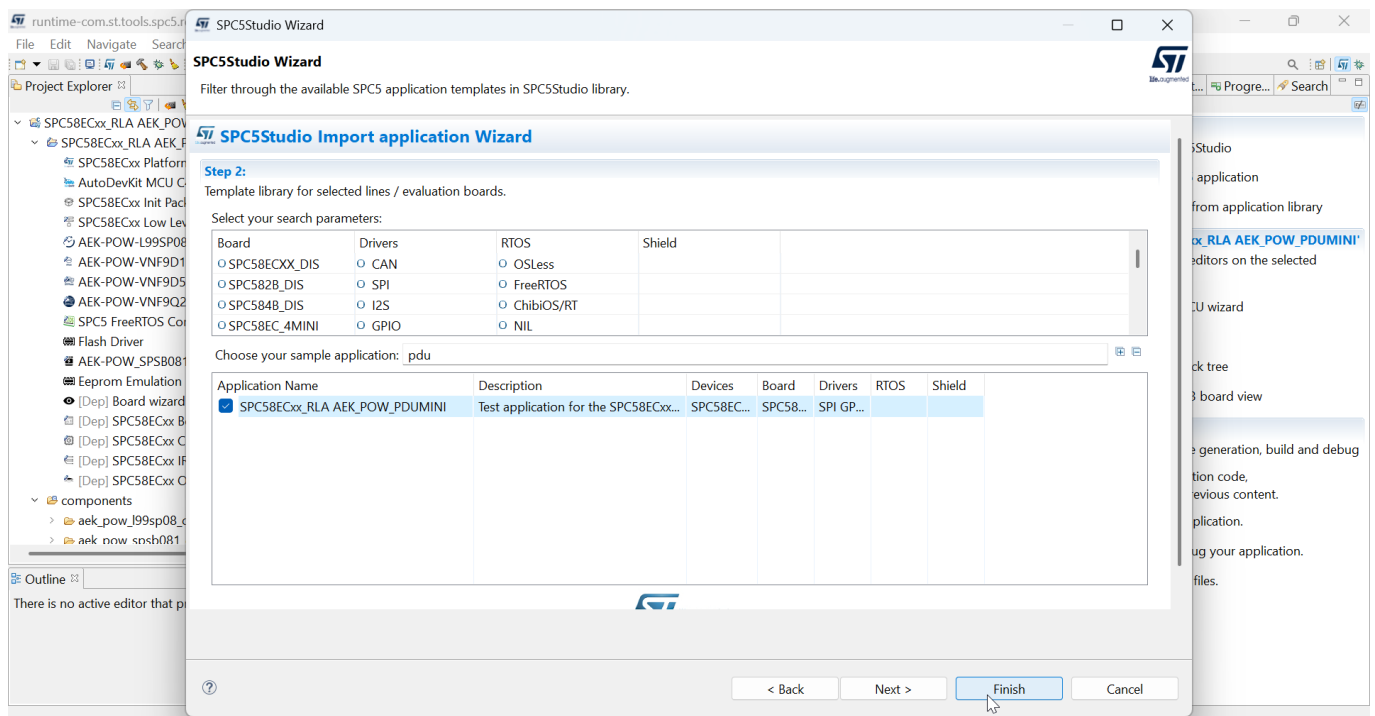
**Step 3.** Type “pdu” in the search bar to find the demo.

**Figure 152. Importing the demo from the application library (3 of 4)**



**Step 4.** Select the demo by ticking the box. Then, click on “Finish”. The demo is ready to be used.

**Figure 153. Importing the demo from the application library (4 of 4)**



## 7 Schematic diagrams

**Figure 154. AEK-POW-PDUMINI circuit schematic (1 of 10)**

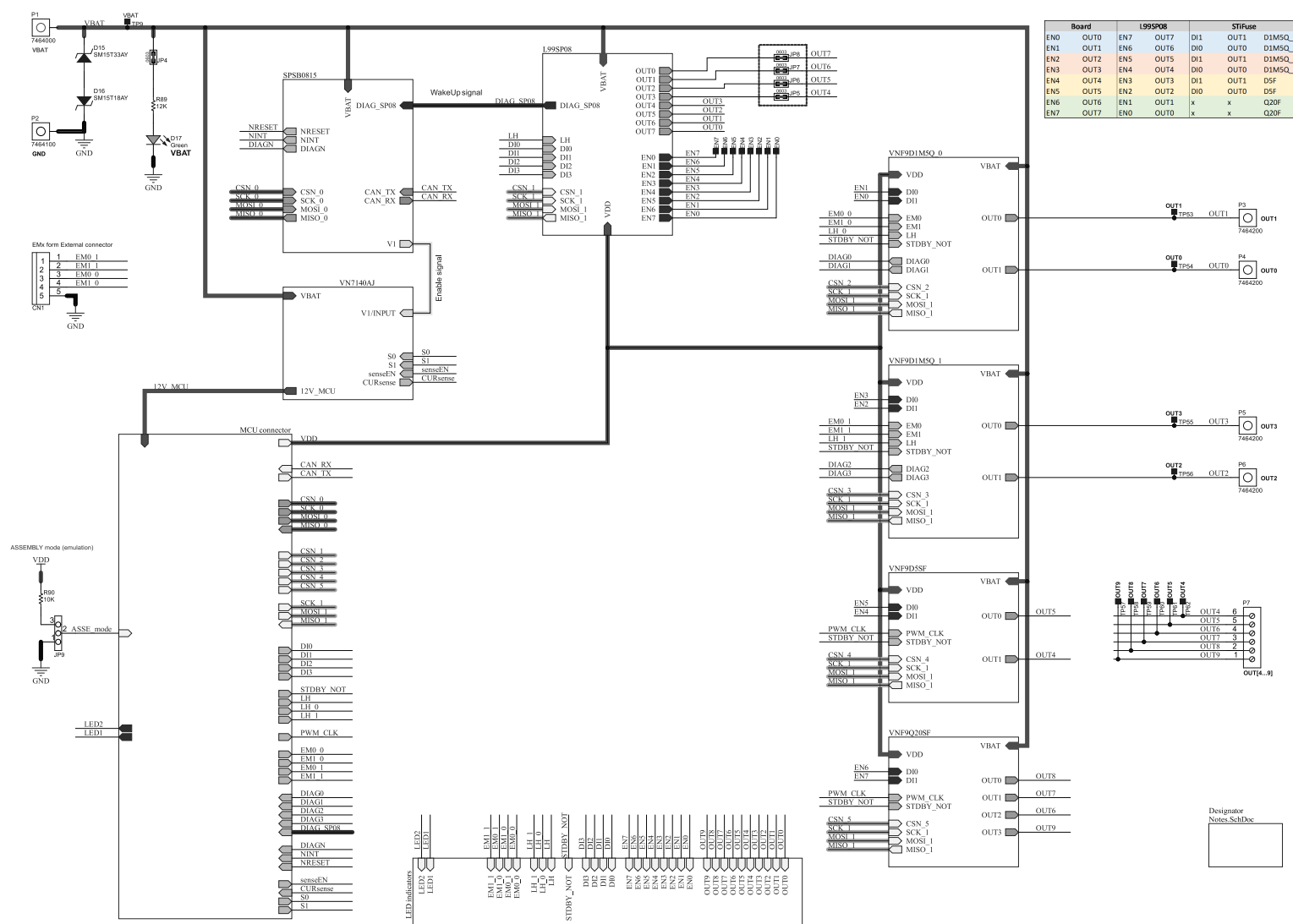


Figure 155. AEK-POW-PDUMINI circuit schematic (2 of 10)

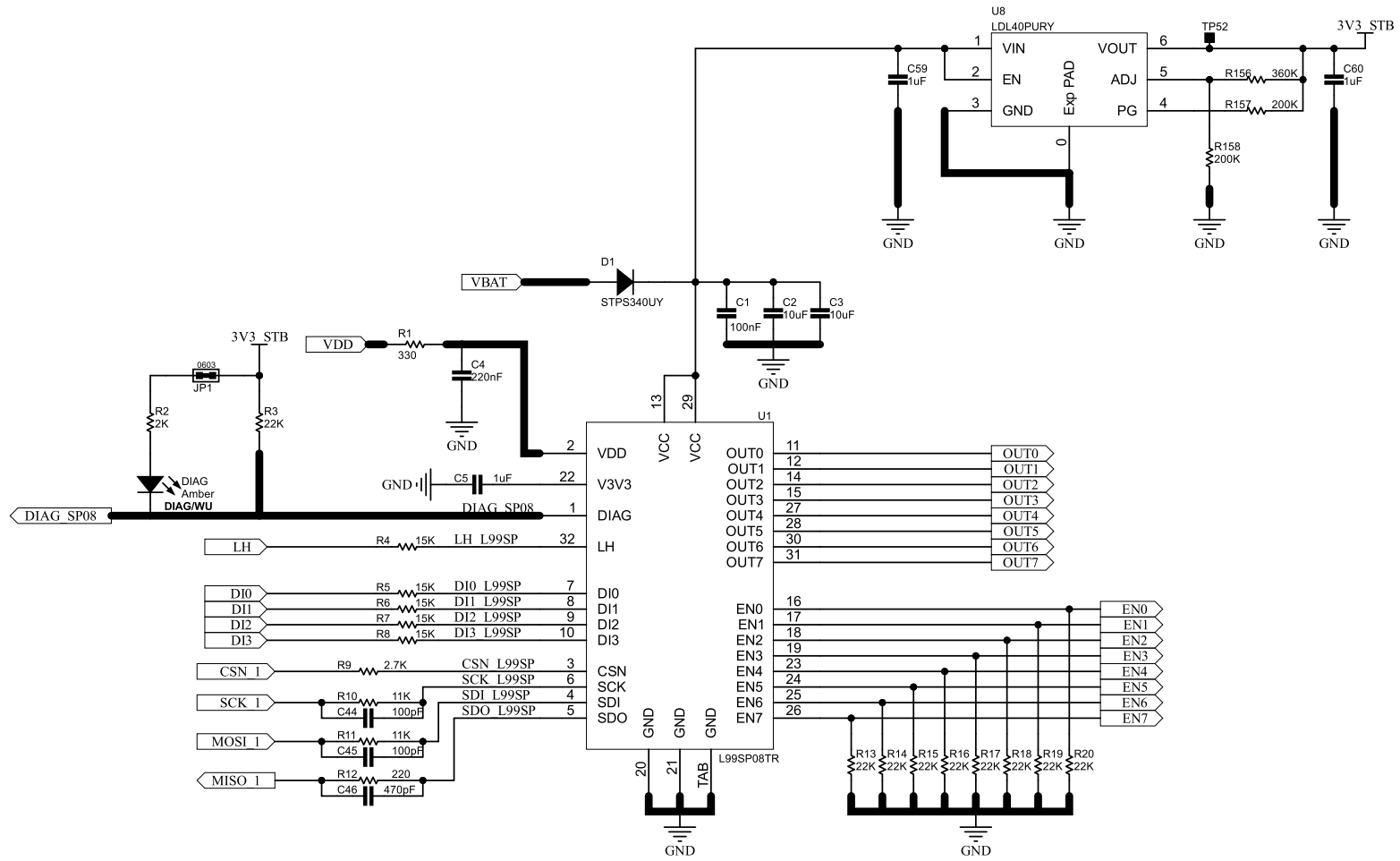
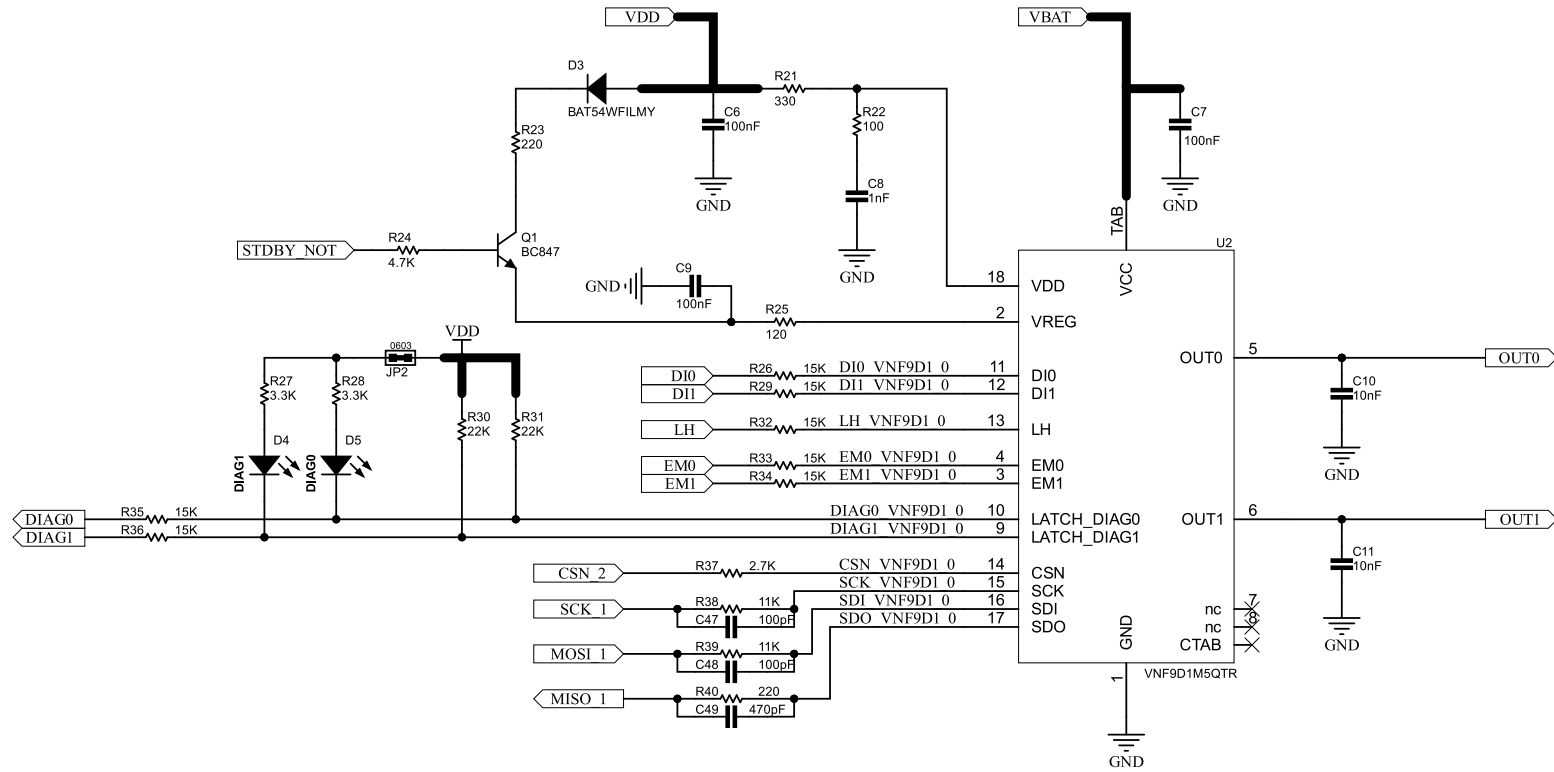




Figure 156. AEK-POW-PDUMINI circuit schematic (3 of 10)



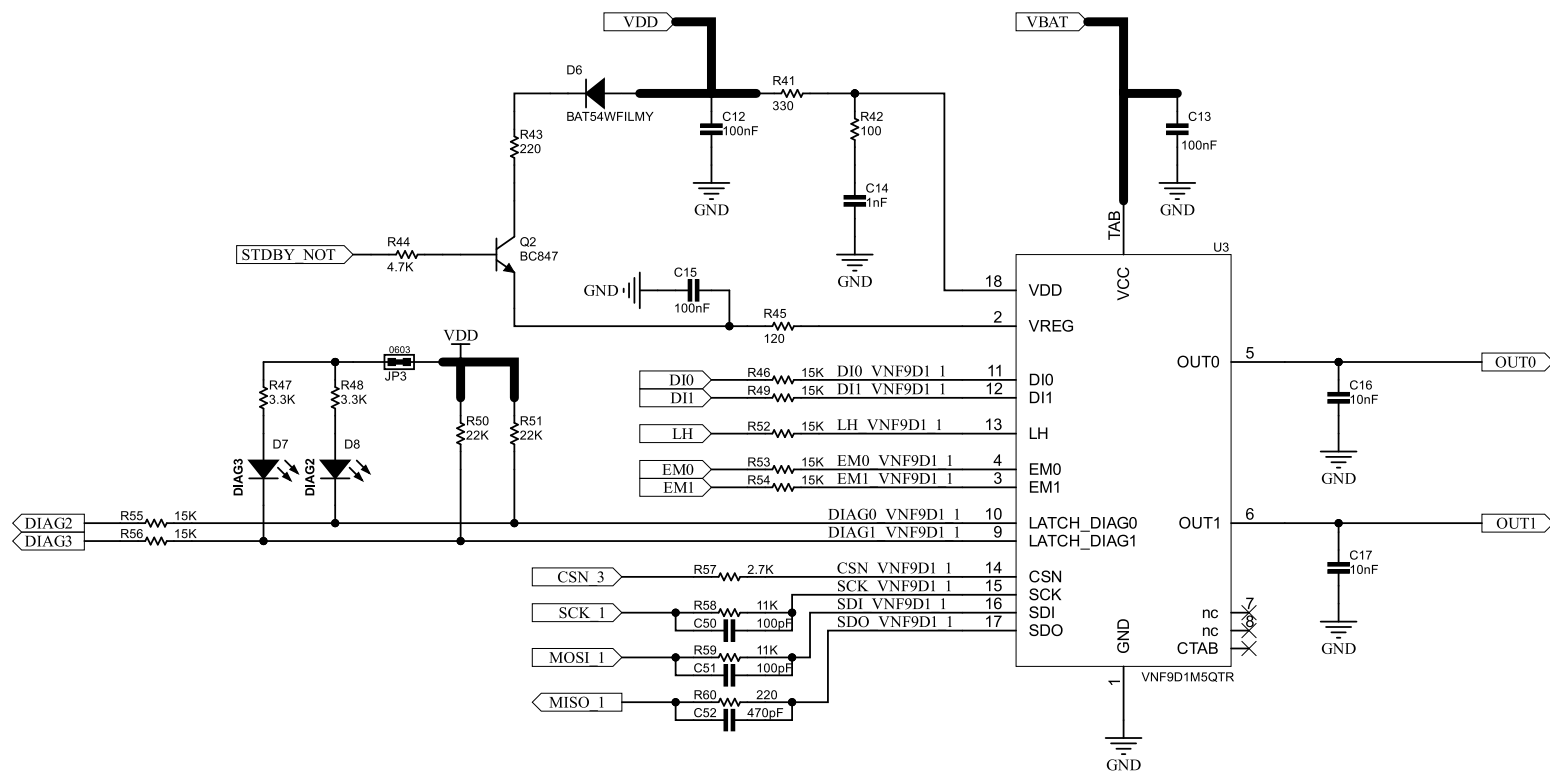


Figure 158. AEK-POW-PDUMINI circuit schematic (5 of 10)

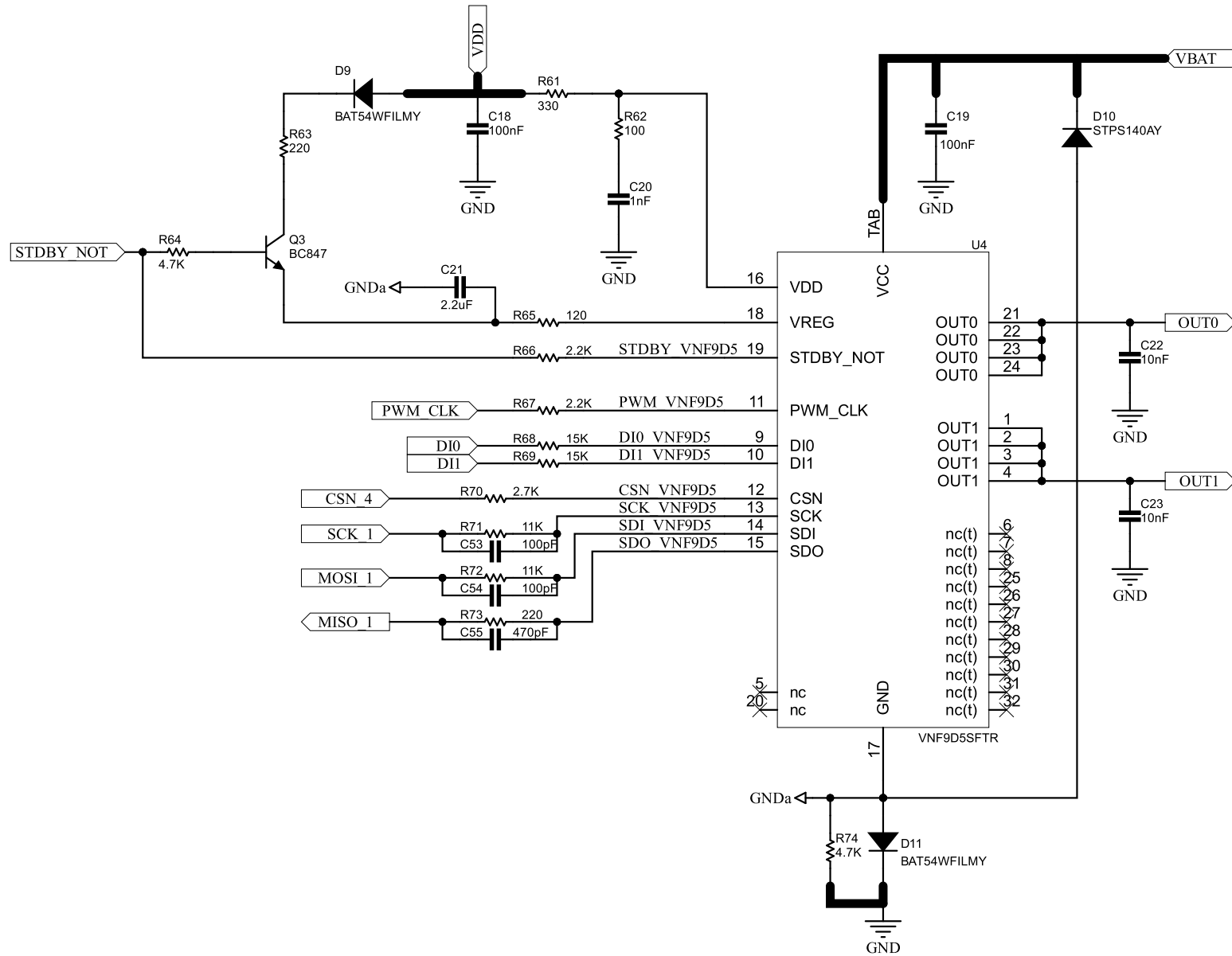
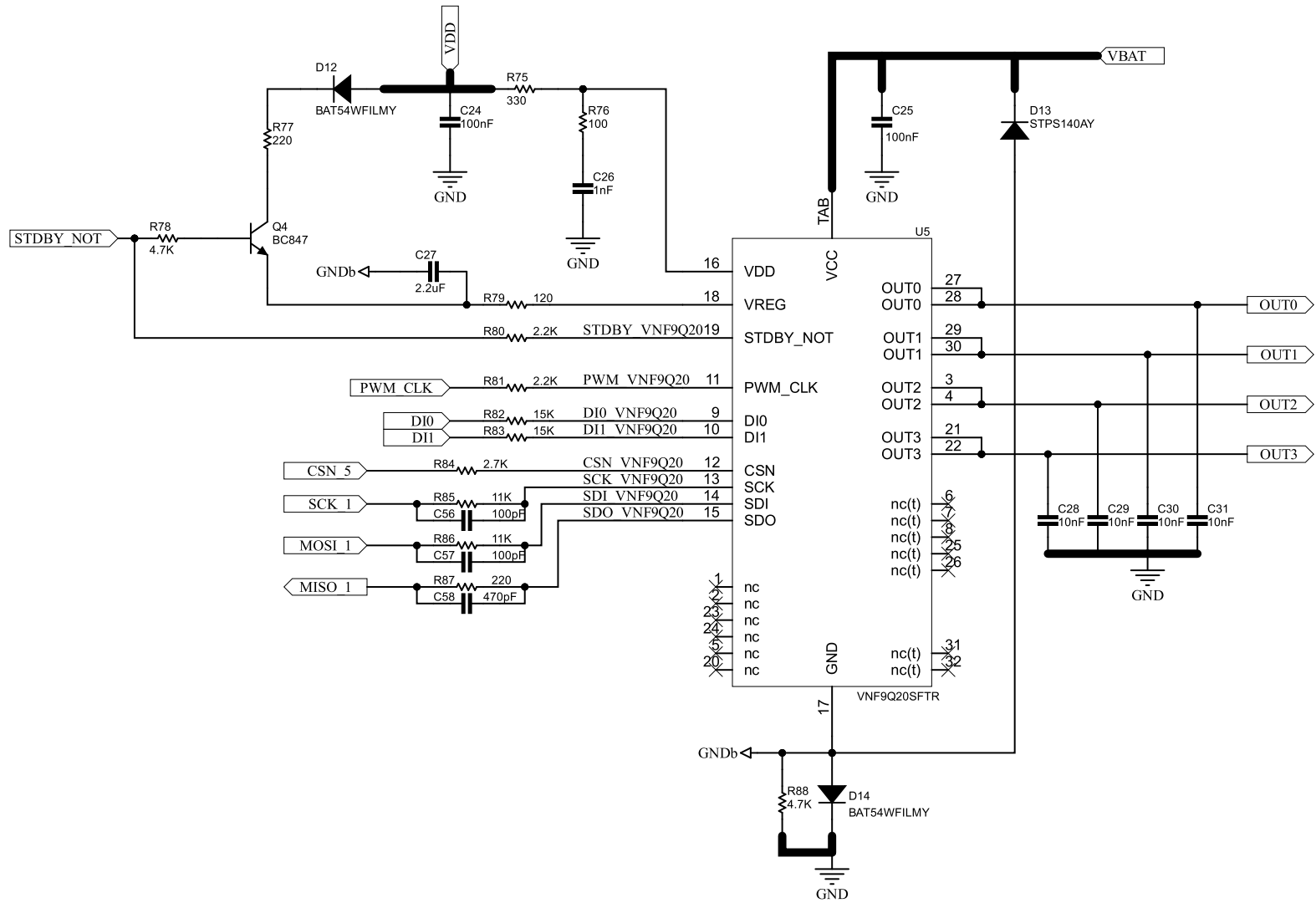


Figure 159. AEK-POW-PDUMINI circuit schematic (6 of 10)



**UM3593**

**Schematic diagrams**

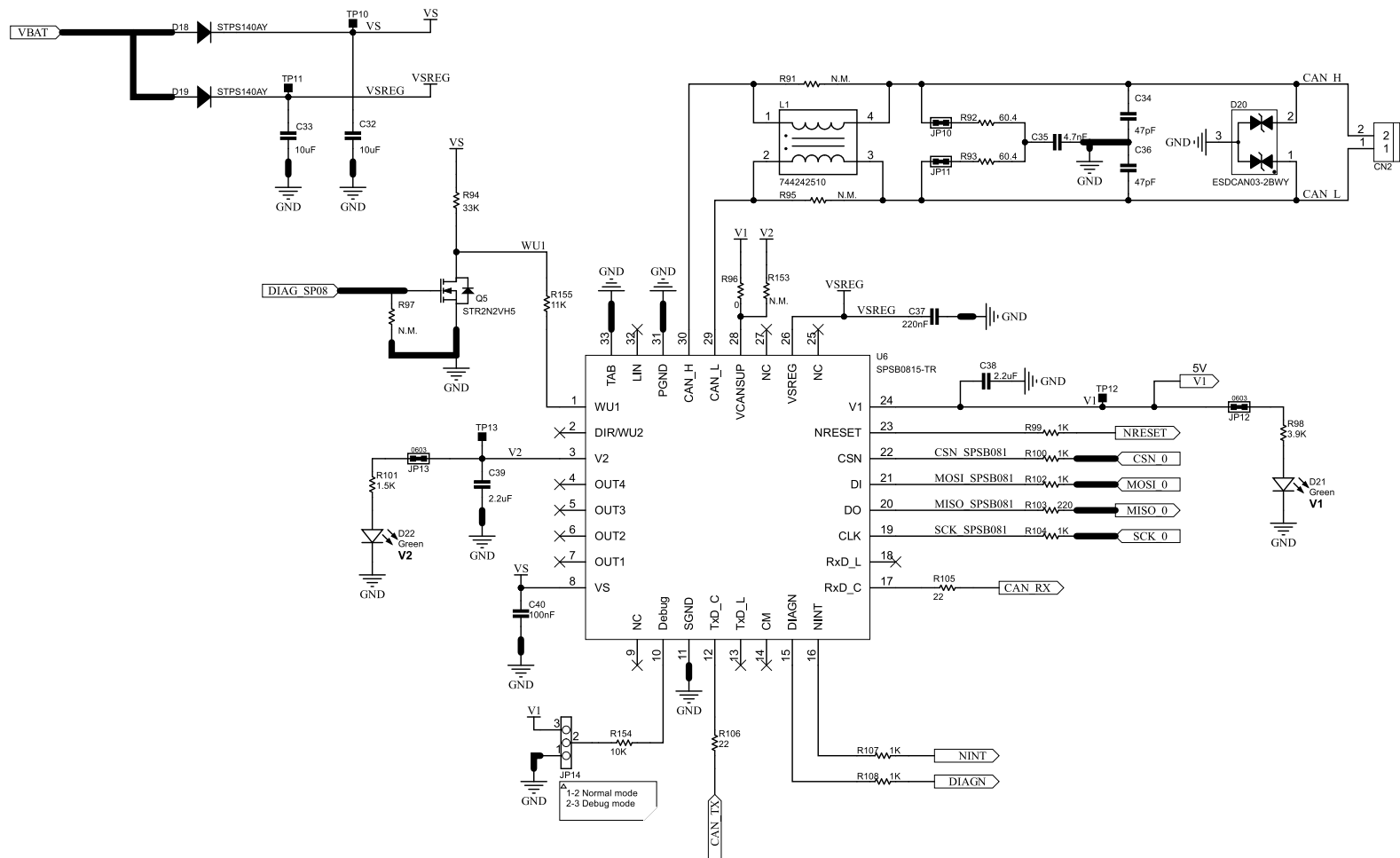




Figure 161. AEK-POW-PDUMINI circuit schematic (8 of 10)

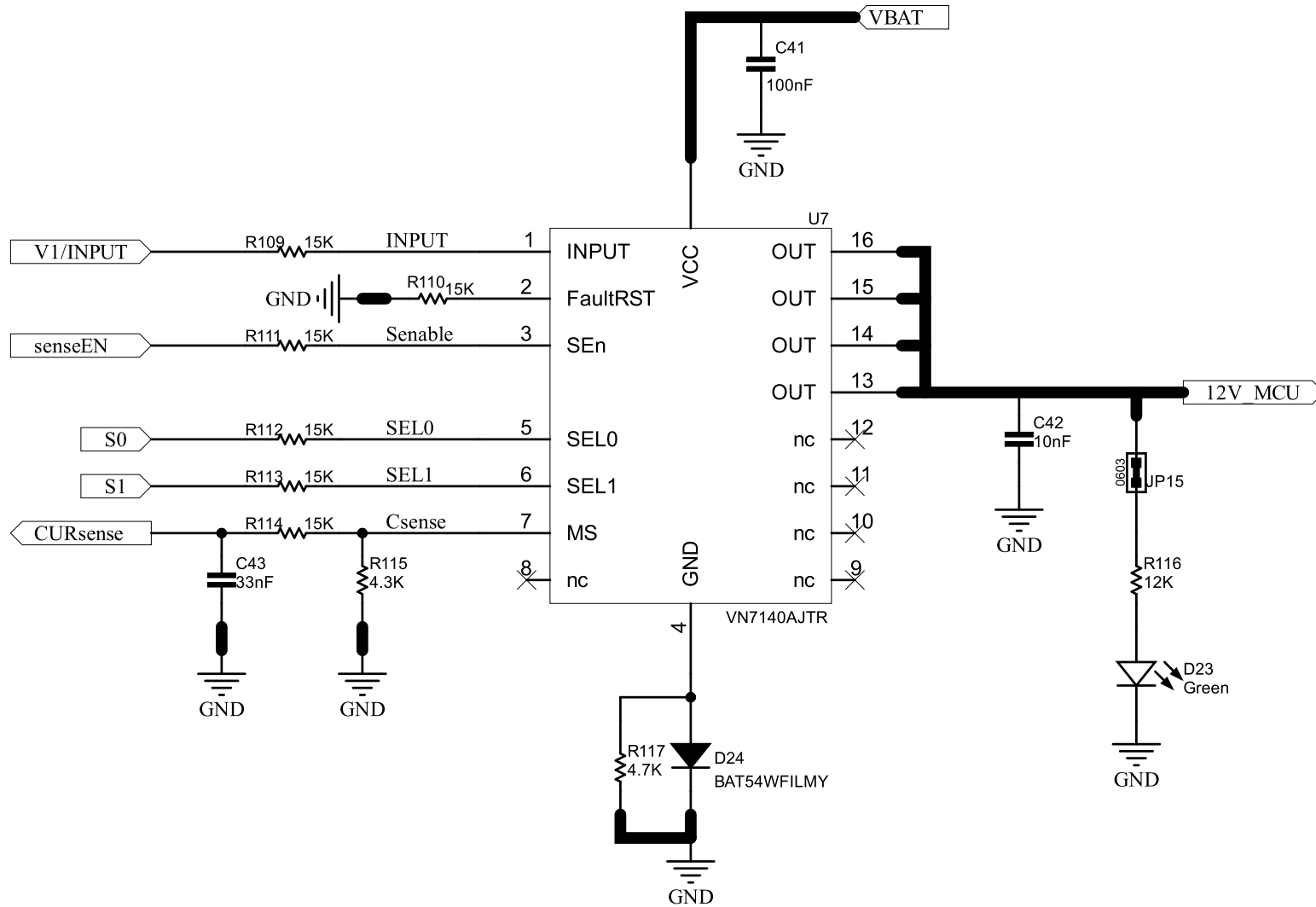


Figure 162. AEK-POW-PDUMINI circuit schematic (9 of 10)

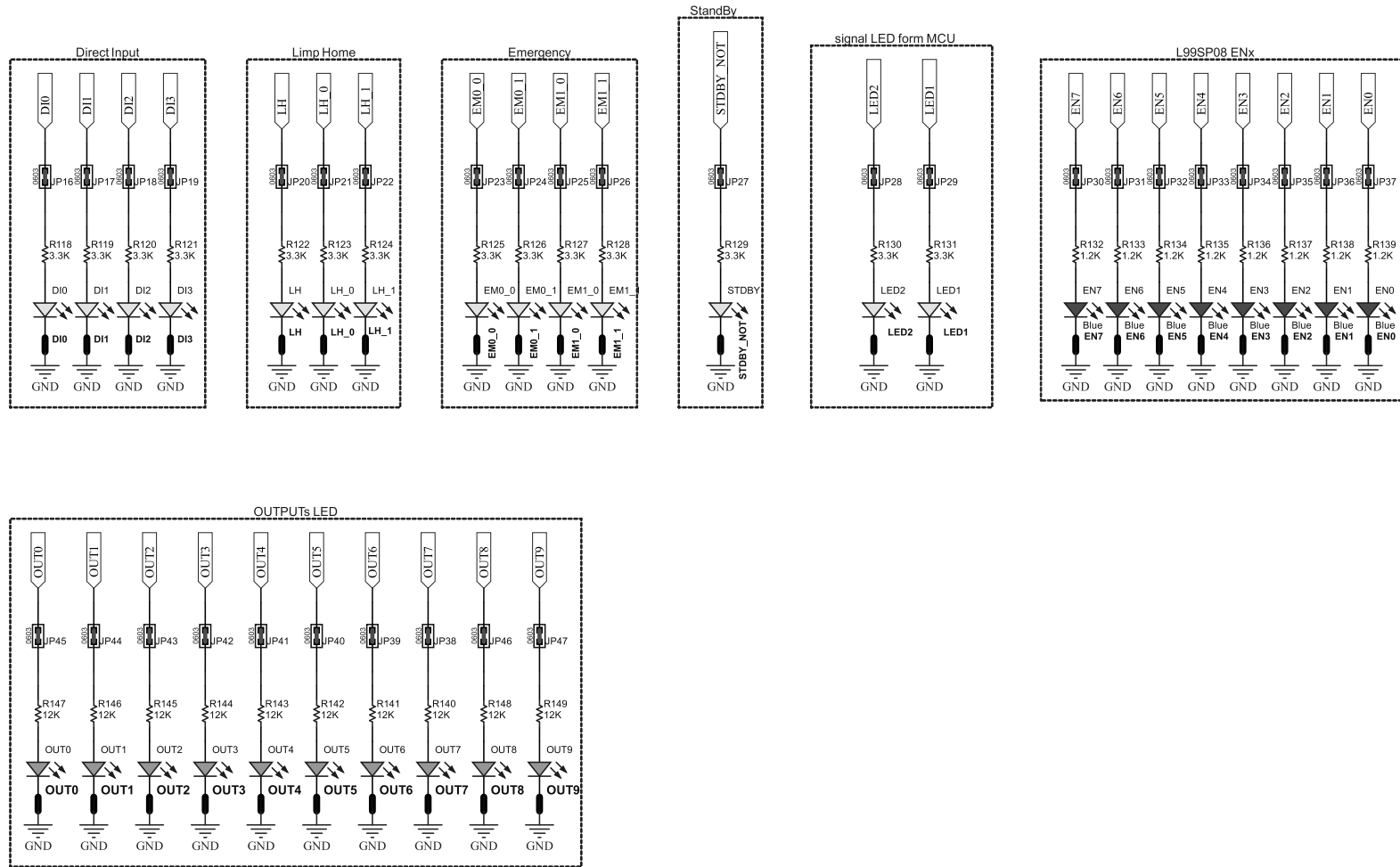
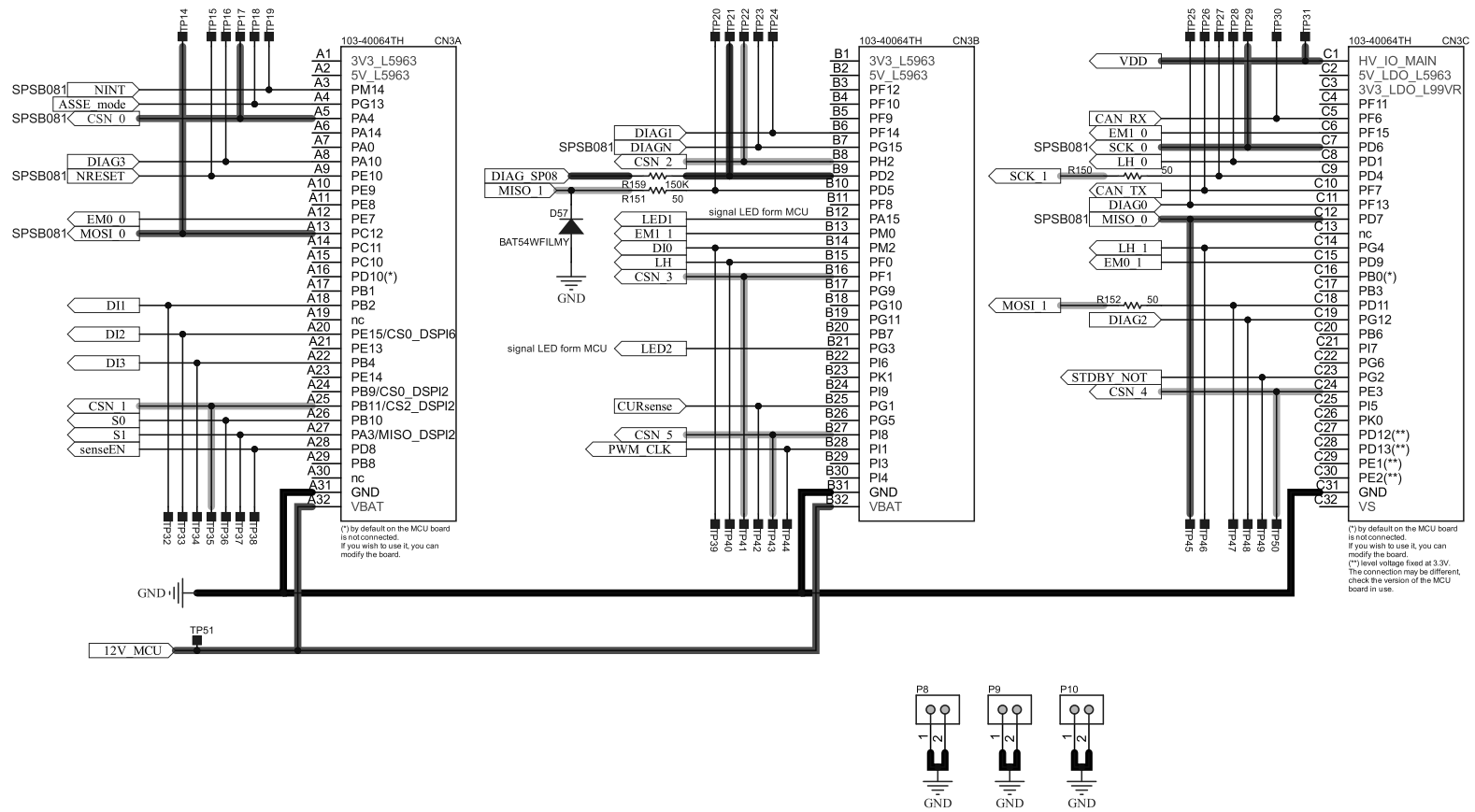


Figure 163. AEK-POW-PDUMINI circuit schematic (10 of 10)



## 8 Bill of materials

**Table 12. AEK-POW-PDUMINI bill of materials**

| Item | Q.ty | Ref.                                                  | Part/value     | Description                                                    | Manufacturer | Order code                |
|------|------|-------------------------------------------------------|----------------|----------------------------------------------------------------|--------------|---------------------------|
| 1    | 6    | C1, C7, C13, C19, C25, C41                            | 100nF          | 0805 - 100V - X7R Class II                                     | WE           | 885012207128              |
| 2    | 4    | C2, C3, C32, C33                                      | 10uF           | 1206 - 50V - X5R Class II, 1206 - 50V - X5R                    | WE           | 885012108022              |
| 3    | 2    | C4, C37                                               | 220nF          | 0603 - 50V - X7R Class II                                      | WE           | 885012206125              |
| 4    | 1    | C5                                                    | 1uF            | 0603 - 25V - X7R Class II                                      | WE           | 885012206076              |
| 5    | 7    | C6, C9, C12, C15, C18, C24, C40                       | 100nF          | 0603 - 50V - X7R Class II                                      | WE           | 885012206095              |
| 6    | 4    | C8, C14, C20, C26                                     | 1nF            | 0603 - 50V - X7R Class II                                      | WE           | 885012206083              |
| 7    | 11   | C10, C11, C16, C17, C22, C23, C28, C29, C30, C31, C42 | 10nF           | 0603 - 50V - X7R Class II                                      | WE           | 885012206089              |
| 8    | 4    | C21, C27, C38, C39                                    | 2.2uF          | 0805 - 25V - X7R Class II                                      | WE           | 885012207079              |
| 9    | 2    | C34, C36                                              | 47pF           | 0603 - 50V - NP0 Class I                                       | WE           | 885012006055              |
| 10   | 1    | C35                                                   | 4.7nF          | 0603 - 50V - X7R Class II                                      | WE           | 885012206087              |
| 11   | 1    | C43                                                   | 33nF           | 0603 - 50V - X7R Class II                                      | WE           | 885012206092              |
| 12   | 10   | C44, C45, C47, C48, C50, C51, C53, C54, C56, C57      | 100pF          | 0603 - 100V - X7R Class II                                     | WE           | 885012206102              |
| 13   | 5    | C46, C49, C52, C55, C58                               | 470pF          | 0603 - 50V - X7R Class II                                      | WE           | 885012206081              |
| 14   | 2    | C59, C60                                              | 1uF            | 0805 - 50V - X7R Class II                                      | WE           | 885012207103              |
| 15   | 1    | CN1                                                   | Con 5P         | 2.54mm - 1 row - KK254 - Male                                  | WE           | 61900511121               |
| 16   | 1    | CN2                                                   | Con 2P         | 2.54mm - 1 row - KK254 - Male                                  | WE           | 61900211121               |
| 17   | 1    | CN3                                                   | 103-40064TH    | DIN 41612 Male 90°, type C, Part No. 103-40064TH               | EPT          | 103-40064TH               |
| 18   | 1    | D1                                                    | STPS340UY, SMB | Automotive power Schottky rectifier - 40V, 3A - Unidirectional | ST           | <a href="#">STPS340UY</a> |

| Item | Q.ty | Ref.                                                                                                                                                                                                                                                                                                                      | Part/value                | Description                                                                       | Manufacturer | Order code    |
|------|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|-----------------------------------------------------------------------------------|--------------|---------------|
| 19   | 8    | D3, D6, D9,<br>D11, D12, D14,<br>D24, D57                                                                                                                                                                                                                                                                                 | BAT54WFILMY,<br>SOT323    | Automotive<br>small signal<br>Schottky diodes<br>- 40V, 300mA -<br>Unidirectional | ST           | BAT54WFILMY   |
| 20   | 5    | D4, D5, D7, D8,<br>DIAG                                                                                                                                                                                                                                                                                                   | Amber                     | 0805 - Led<br>Amber - 2V                                                          | WE           | 150080AS75000 |
| 21   | 4    | D10, D13, D18,<br>D19                                                                                                                                                                                                                                                                                                     | STPS140AY,<br>SMA         | Automotive<br>power Schottky<br>rectifier, 40V 1A                                 | ST           | STPS140AY     |
| 22   | 1    | D15                                                                                                                                                                                                                                                                                                                       | SM15T33AY,<br>SMC         | Automotive<br>1500 W TVS in<br>SMC - 33VDC -<br>Unidirectional                    | ST           | SM15T33AY     |
| 23   | 1    | D16                                                                                                                                                                                                                                                                                                                       | SM15T18AY,<br>SMC         | Automotive<br>1500 W TVS in<br>SMC - 18VDC -<br>Unidirectional                    | ST           | SM15T18AY     |
| 24   | 14   | D17, D21, D22,<br>D23, OUT0,<br>OUT1, OUT2,<br>OUT3, OUT4,<br>OUT5, OUT6,<br>OUT7, OUT8,<br>OUT9                                                                                                                                                                                                                          | Green                     | 0805 - Led<br>Green - 3.2V                                                        | WE           | 150080GS75000 |
| 25   | 1    | D20                                                                                                                                                                                                                                                                                                                       | ESDCAN03-2B<br>WY, SOT323 | Automotive<br>dual-line TVS in<br>SOT323-3L for<br>CAN bus                        | ST           | ESDCAN03-2BWY |
| 26   | 14   | DI0, DI1, DI2,<br>DI3, EM0_0,<br>EM0_1, EM1_0,<br>EM1_1, LED1,<br>LED2, LH,<br>LH_0, LH_1,<br>STDBY                                                                                                                                                                                                                       | Yellow                    | 0805 - Led<br>Yellow - 2V                                                         | WE           | 150080YS75000 |
| 27   | 8    | EN0, EN1, EN2,<br>EN3, EN4, EN5,<br>EN6, EN7                                                                                                                                                                                                                                                                              | Blue                      | 0805 - Led Blue<br>- 3.2V                                                         | WE           | 150080BS75000 |
| 28   | 44   | JP1, JP2, JP3,<br>JP4, JP5, JP6,<br>JP7, JP8, JP12,<br>JP13, JP15,<br>JP16, JP17,<br>JP18, JP19,<br>JP20, JP21,<br>JP22, JP23,<br>JP24, JP25,<br>JP26, JP27,<br>JP28, JP29,<br>JP30, JP31,<br>JP32, JP33,<br>JP34, JP35,<br>JP36, JP37,<br>JP38, JP39,<br>JP40, JP41,<br>JP42, JP43,<br>JP44, JP45,<br>JP46, JP47,<br>R96 | 0                         | 0603_Jumper<br>2p Closed SMD,<br>0603 - $\pm 1\%$ -<br>0.1W                       | Panasonic    | ERJ3GEY0R00V  |



| Item | Q.ty | Ref.                   | Part/value        | Description                                                                              | Manufacturer | Order code                |
|------|------|------------------------|-------------------|------------------------------------------------------------------------------------------|--------------|---------------------------|
| 29   | 2    | JP9, JP14              | Jumper 3p         | THT Vertical 3 pins Header, Pitch 2.54 mm, Single Row                                    | WE           | 61300311121               |
| 30   | 2    | JP10, JP11             | Jumper 2p Closed  | 2mm - WR-PHD 2.00 mm THT Pin Header Single Row, Vertical, 2p, Closed                     | WE           | 62000211121               |
| 31   | 1    | L1                     | 51uH              | WE-SLM SMT Common Mode Line Filter - 51uH                                                | WE           | 744242510                 |
| 32   | 1    | P1                     | 7464000           | WP-PLUG REDCUBE Direct Plug Terminal, L14.3W14.3H18 .1mm, Red, 120A                      | WE           | 7464000                   |
| 33   | 1    | P2                     | 7464100           | WP-PLUG REDCUBE Direct Plug Terminal, L14.3W14.3H18 .1mm, Black, 120A                    | WE           | 7464100                   |
| 34   | 4    | P3, P4, P5, P6         | 7464200           | WP-PLUG REDCUBE Direct Plug Terminal, L14.3W14.3H18 .1mm, Blue, 120A                     | WE           | 7464200                   |
| 35   | 1    | P7                     | Con 6p 5.08_green | WR-TBL Series 2425 Horizontal Entry Modular, THT, pitch 5.08mm, 06p                      | WE           | 691242510006              |
| 36   | 3    | P8, P9, P10            | Header 2p 2.54    | 2.54mm - WR-PHD Pin Header, THT, pitch 2.54mm, Single Row, Vertical, 2p                  | WE           | 61300211121               |
| 37   | 4    | Q1, Q2, Q3, Q4         | BC847             | NPN Bipolar Transistor 45V 100mA                                                         | Nexperia     | BC847                     |
| 38   | 1    | Q5                     | STR2N2VH5, SOT-23 | N-channel 20 V, 0.025 $\Omega$ typ., 2.3 A STripFET™ H5 Power MOSFET in a SOT-23 package | ST           | <a href="#">STR2N2VH5</a> |
| 39   | 5    | R1, R21, R41, R61, R75 | 330               | 0603 $\pm 1\%$ - 0.125W                                                                  | Panasonic    | ERJ-H3EF3300V             |
| 40   | 1    | R2                     | 2K                | 0603 $\pm 1\%$ - 0.2W                                                                    | Panasonic    | ERJP03F2001V              |

| Item | Q.ty | Ref.                                                                                                                                             | Part/value | Description               | Manufacturer | Order code         |
|------|------|--------------------------------------------------------------------------------------------------------------------------------------------------|------------|---------------------------|--------------|--------------------|
| 41   | 13   | R3, R13, R14, R15, R16, R17, R18, R19, R20, R30, R31, R50, R51                                                                                   | 22K        | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJ-UP3F2202V      |
| 42   | 29   | R4, R5, R6, R7, R8, R26, R29, R32, R33, R34, R35, R36, R46, R49, R52, R53, R54, R55, R56, R68, R69, R82, R83, R109, R110, R111, R112, R113, R114 | 15K        | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJ-PA3F1502V      |
| 43   | 5    | R9, R37, R57, R70, R84                                                                                                                           | 2.7K       | 0603 - $\pm 1\%$ - 0.1W   | Vishay       | MCT06030C2701FP500 |
| 44   | 11   | R10, R11, R38, R39, R58, R59, R71, R72, R85, R86, R155                                                                                           | 11K        | 0603 - $\pm 1\%$ - 0.33W  | Panasonic    | ERJ-PA3F1102V      |
| 45   | 10   | R12, R23, R40, R43, R60, R63, R73, R77, R87, R103                                                                                                | 220        | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJPA3F2200V       |
| 46   | 4    | R22, R42, R62, R76                                                                                                                               | 100        | 0603 - $\pm 1\%$ - 0.125W | Panasonic    | ERJH3EF1000V       |
| 47   | 7    | R24, R44, R64, R74, R78, R88, R117                                                                                                               | 4.7K       | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJPA3F4701V       |
| 48   | 4    | R25, R45, R65, R79                                                                                                                               | 120        | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJPA3F1200V       |
| 49   | 18   | R27, R28, R47, R48, R118, R119, R120, R121, R122, R123, R124, R125, R126, R127, R128, R129, R130, R131                                           | 3.3K       | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJ-PA3F3301V      |
| 50   | 4    | R66, R67, R80, R81                                                                                                                               | 2.2K       | 0603 - $\pm 1\%$ - 0.1W   | Panasonic    | ERJ-3EKF2201V      |
| 51   | 12   | R89, R116, R140, R141, R142, R143, R144, R145, R146, R147, R148, R149                                                                            | 12K        | 0603 - $\pm 1\%$ - 0.1W   | Panasonic    | ERJ-3EKF1202V      |
| 52   | 2    | R90, R154                                                                                                                                        | 10K        | 0603 - $\pm 1\%$ - 0.2W   | Panasonic    | ERJP03F1002V       |
| 53   | 3    | R91, R95, R97                                                                                                                                    | N.M.       | 603                       | N.A.         | N.A.               |
| 54   | 2    | R92, R93                                                                                                                                         | 60.4       | 0603 - $\pm 1\%$ - 0.1W   | Panasonic    | ERJ3EKF60R4V       |
| 55   | 1    | R94                                                                                                                                              | 33K        | 0603 - $\pm 1\%$ - 0.25W  | Panasonic    | ERJ-PA3F3302V      |

| Item | Q.ty | Ref.                                                    | Part/value                  | Description                                                                                                                                      | Manufacturer | Order code                  |
|------|------|---------------------------------------------------------|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--------------|-----------------------------|
| 56   | 1    | R98                                                     | 3.9K                        | 0603 - $\pm 1\%$ - 0.1W                                                                                                                          | Panasonic    | ERJ3EKF3901V                |
| 57   | 6    | R99, R100,<br>R102, R104,<br>R107, R108                 | 1K                          | 0603 - $\pm 1\%$ - 0.25W                                                                                                                         | Panasonic    | ERJPA3F1001V                |
| 58   | 1    | R101                                                    | 1.5K                        | 0603 - $\pm 1\%$ - 0.25W                                                                                                                         | Panasonic    | ERJ-PA3F1501V               |
| 59   | 2    | R105, R106                                              | 22                          | 0603 - $\pm 1\%$ - 0.125W                                                                                                                        | Panasonic    | ERJH3EF22R0V                |
| 60   | 1    | R115                                                    | 4.3K                        | 0603 - $\pm 1\%$ - 0.1W                                                                                                                          | Panasonic    | ERJ-3EKF4301V               |
| 61   | 8    | R132, R133,<br>R134, R135,<br>R136, R137,<br>R138, R139 | 1.2K                        | 0603 - $\pm 0.5\%$ - 0.25W                                                                                                                       | Panasonic    | ERJ-UP3D1201V               |
| 62   | 3    | R150, R151,<br>R152                                     | 50                          | 0603 - $\pm 1\%$ - 0.100W                                                                                                                        | Vishay       | CRCW060350R0FKTA            |
| 63   | 1    | R153                                                    | N.M.                        | 603                                                                                                                                              | N.A.         | N.A.                        |
| 64   | 1    | R156                                                    | 360K                        | 0603 - $\pm 1\%$ - 0.1W                                                                                                                          | Panasonic    | ERJ-3EKF3603V               |
| 65   | 2    | R157, R158                                              | 200K                        | 0603 - $\pm 1\%$ - 0.1W                                                                                                                          | Panasonic    | ERJ-3EKF2003V               |
| 66   | 1    | R159                                                    | 150K                        | 0603 - $\pm 5\%$ - 0.33W                                                                                                                         | Panasonic    | ERJ-PA3J154V                |
| 67   | 1    | U1                                                      | L99SP08TR,<br>Tape and Reel | Smart octa P-channel for ultra-low current consumption automotive systems, Ron 1 $\Omega$                                                        | ST           | <a href="#">L99SP08TR</a>   |
| 68   | 2    | U2, U3                                                  | VNF9D1M5QTR                 | 2-channel high-side driver with STi <sup>2</sup> Fuse protection for automotive power distribution applications, 27A and RON typ. 1.5 m $\Omega$ | ST           | <a href="#">VNF9D1M5QTR</a> |
| 69   | 1    | U4                                                      | VNF9D5SFTR                  | 2-channel high-side driver with STi <sup>2</sup> Fuse protection for automotive power distribution applications, 15A and RON typ. 5.9 m $\Omega$ | ST           | <a href="#">VNF9D5SFTR</a>  |

| Item | Q.ty | Ref.        | Part/value                               | Description                                                                                                                              | Manufacturer | Order code     |
|------|------|-------------|------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|--------------|----------------|
| 70   | 1    | U5          | VNF9Q20SFTR                              | 4-channel high-side driver with STi <sup>2</sup> Fuse protection for automotive power distribution applications, 6A and RON typ. 21.5 mΩ | ST           | VNF9Q20SFTR    |
| 71   | 1    | U6          | SPSB0815-TR, QFN-32L WF                  | Automotive Power management IC with LIN and CAN-FD                                                                                       | ST           | SPSB0815-TR    |
| 72   | 1    | U7          | VN7140AJTR, PowerSSO 16                  | High-side driver with MultiSense analog feedback for automotive applications, 28V and RON typ. 140 mΩ                                    | ST           | VN7140AJTR     |
| 73   | 1    | U8          | LDL40PURY, DFN 6L 2X2X 0.75 PITCH0.65 AG | 200 mA low dropout LDO                                                                                                                   | ST           | LDL40PURY      |
| 74   | 2    | for blister | 60900213421                              | WR-PHD 2.54 mm Multi-Jumper Jumper with Test Point                                                                                       | WE           | 60900213421    |
| 75   | 4    | for blister | 970150365                                | WA-SPAIL Plastic Spacer Stud, metric, internal/ internal                                                                                 | WE           | 970150365      |
| 76   | 4    | for blister | 97790603211                              | WA-SCRW Pan Head Screw w. cross slot M3                                                                                                  | WE           | 97790603211    |
| 77   | 1    | for blister | 61900211621                              | WR-WTB 2.54 mm Female Terminal Housing                                                                                                   | WE           | 61900211621    |
| 78   | 1    | for blister | 61900511621                              | WR-WTB 2.54 mm Female Terminal Housing                                                                                                   | WE           | 61900511621    |
| 79   | 8    | for blister | 61900113722D EC                          | WR-WTB 2.54 mm Female Crimp Contact                                                                                                      | WE           | 61900113722DEC |
| 80   | 2    | for blister | 60800213421                              | WR-PHD 2.00 mm Multi-Jumper Jumper with Test Point                                                                                       | WE           | 60800213421    |
| 81   | 2    | for blister | 74640016                                 | WP-PLUG REDCUBE Cable Connector for Direct PLUG Terminal                                                                                 | WE           | 74640016       |

| Item | Q.ty | Ref.        | Part/value | Description                                                             | Manufacturer | Order code |
|------|------|-------------|------------|-------------------------------------------------------------------------|--------------|------------|
| 82   | 4    | for blister | 7464006    | WP-PLUG<br>REDCUBE<br>Cable<br>Connector for<br>Direct PLUG<br>Terminal | WE           | 7464006    |



## 9 Board versions

**Table 13. AEK-POW-PDUMINI versions**

| PCB version                      | Schematic diagrams                                   | Bill of materials                                   |
|----------------------------------|------------------------------------------------------|-----------------------------------------------------|
| AEK\$POW-PDUMINIA <sup>(1)</sup> | <a href="#">AEK\$POW-PDUMINIA schematic diagrams</a> | <a href="#">AEK\$POW-PDUMINIA bill of materials</a> |

1. This code identifies the AEK-POW-PDUMINI evaluation board first version. It is printed on the board PCB.

## 10 Regulatory compliance information

### Notice for US Federal Communication Commission (FCC)

For evaluation only; not FCC approved for resale

FCC NOTICE - This kit is designed to allow:

- (1) Product developers to evaluate electronic components, circuitry, or software associated with the kit to determine whether to incorporate such items in a finished product and
- (2) Software developers to write software applications for use with the end product.

This kit is not a finished product and when assembled may not be resold or otherwise marketed unless all required FCC equipment authorizations are first obtained. Operation is subject to the condition that this product not cause harmful interference to licensed radio stations and that this product accept harmful interference. Unless the assembled kit is designed to operate under part 15, part 18 or part 95 of this chapter, the operator of the kit must operate under the authority of an FCC license holder or must secure an experimental authorization under part 5 of this chapter 3.1.2.

### Notice for Innovation, Science and Economic Development Canada (ISED)

For evaluation purposes only. This kit generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to Industry Canada (IC) rules.

À des fins d'évaluation uniquement. Ce kit génère, utilise et peut émettre de l'énergie radiofréquence et n'a pas été testé pour sa conformité aux limites des appareils informatiques conformément aux règles d'Industrie Canada (IC).

### Notice for the European Union

This device is in conformity with the essential requirements of the Directive 2014/30/EU (EMC) and of the Directive 2011/65/EU (RoHS II), including subsequent revisions and additions, as well as amended by the Delegated Directive 2015/863/EU (RoHS III).

### Notice for the United Kingdom

This device is in conformity with the relevant provisions of the following EU legislation recognized by the United Kingdom:

- Electromagnetic Compatibility Regulations 2016 (S.I. 2016/1091) implementing Directive 2014/30/EU (EMC)
- The Restriction of the Use of Certain Hazardous Substances in Electrical and Electronic Equipment Regulations 2012 (RoHS) (S.I. 2012/3032) implementing Directive 2011/65/EU (RoHS II), including subsequent revisions and additions, as well as amended by the Delegated Directive 2015/863/EU (RoHS III).

## Appendix A UDS extensions and threshold configuration

This appendix describes the UDS extensions and the internal EFUSE service layer used to configure current thresholds and I<sup>2</sup>t protection parameters for the following devices:

- L99SP08
  - VNF9D1M5Q
  - VNF9D5SF
  - VNF9Q20SF
- In detail, the appendix focuses on:
- The added UDS Data Identifiers (DIDs) and the expected request formats.
  - The internal EFUSE service IDs and their role.
  - The mapping between PDU channels, device channels and EEPROM layout.
  - The power-on initialization logic that loads configuration from EEPROM or applies default configurations.

### A.1 Supported SID

The configuration services use the ISO 14229-1 standard service (SID 0x2E – WriteDataByIdentifier), used to write:

- L99SP08 current threshold ( $I_{THR}$ ).
- I<sup>2</sup>t configuration ( $I_{NOM}$ ,  $T_{NOM}$ ,  $I_{PEAK}/I_{SHUTOFF}$ ) of VNFD devices.

### A.2 Data identifiers added to the existing set

In Unified Diagnostic Services, a DID identifies a specific diagnostic data item accessible through diagnostic services.

The following Data Identifiers (DIDs) have been added to the existing UDS DID list.

- 0xF12A – L99SP08 current threshold  $I_{THR}$   
Provides the current threshold parameter of the L99SP08 device.
- 0xF12B – VNFD1M5Q I<sup>2</sup>t parameters  
Provides the I<sup>2</sup>T protection parameters of the VNFD1M5Q device:  $I_{NOM}$ ,  $T_{NOM}$ , and  $I_{PEAK}$ .
- 0xF12C – VNF9D5SF I<sup>2</sup>t parameters  
Provides the I<sup>2</sup>t protection parameters of the VNF9D5SF device:  $I_{NOM}$ ,  $T_{NOM}$ , and  $I_{SHUTOFF}$ .
- 0xF12D – VNF9Q20SF I<sup>2</sup>t parameters  
Provides the I<sup>2</sup>t protection parameters of the VNF9Q20SF device:  $I_{NOM}$ ,  $T_{NOM}$ , and  $I_{SHUTOFF}$ .

### A.3 Request format

The generic UDS request format for these DIDs is as follows:

Byte 0: LEN (number of UDS bytes after this length)

Byte 1: SID = 0x2E (WriteDataByIdentifier)

Byte 2: DID High

Byte 3: DID Low

Note: The DIDs listed in the previous section are split into: DID HIGH and DID low. For example, 0xF12A becomes DID\_HIGH: F1 and DID\_LOW: 2A

Byte 4: CHANNEL (EFUSE\_sev\_channel\_t)

Byte 5: PARAM\_0 (e.g.  $I_{THR}$  or  $I_{NOM}$ )

Byte 6: PARAM\_1 (e.g.  $T_{NOM}$ )

Byte 7: PARAM\_2 (e.g.  $I_{PEAK}$  or  $I_{SHUTOFF}$ )

The actual number of parameter bytes depends on the DID:

- CHANNEL\_ITHR\_L99SP08: uses only CHANNEL and PARAM\_0 (I\_THR).
- I²t DIDs use:

CHANNEL, PARAM\_0 (INOM), PARAM\_1 (TNOM), PARAM\_2 (IPEAK or ISHUTOFF).

## A.4 Channel mapping (AEK-POW-PDUMINI ↔ devices)

Each AEK-POW-PDUMINI output channel can correspond to different physical channels on different power devices.

The UDS and EFUSE layers use logical PDU channels (EFUSE\_OUT0...EFUSE\_OUT9), while each device driver (L99SP08, VNF9D1M5Q, VNF9D5SF, VNF9Q20SF) uses its own channel numbering and register layout.

This section details how AEK-POW-PDUMINI channels, EFUSE channels, and device channels have been mapped and how each EFUSE channel is routed to the corresponding device channel and EEPROM indexes, using the same EFUSE channel enumeration across all services.

### A.4.1 Channel mapping

Almost all physical channels of the AEK-POW-PDUMINI correspond to both an L99SP08 channel and a VNFxxxx channel when these channels are placed in parallel to each other.

Only output 8 and output 9 correspond to two VNF9Q20SF channels without an L99SP08 channel in between.

The following table shows the channel mapping between devices and AEK-POW-PDUMINI channels.

**Table 14. Mapping between onboard devices and AEK-POW-PDUMINI channels**

| Device    | AEK-POW-PDUMINI output channel | Device channel            | Notes                                                                                                                                                                                                            |
|-----------|--------------------------------|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| L99SP08   | EFUSE_OUT0                     | L99SP08_CH7               | This mapping is implemented in <i>EFUSE_sev_setChannelThresholdL99(...)</i> by switching <i>EFUSE_sev_channel</i> on and calling <i>L99SP08_ch_setThreshold(...)</i> through the appropriate <i>L99SP08_CH</i> . |
|           | EFUSE_OUT1                     | L99SP08_CH6               |                                                                                                                                                                                                                  |
|           | EFUSE_OUT2                     | L99SP08_CH5               |                                                                                                                                                                                                                  |
|           | EFUSE_OUT3                     | L99SP08_CH4               |                                                                                                                                                                                                                  |
|           | EFUSE_OUT4                     | L99SP08_CH3               |                                                                                                                                                                                                                  |
|           | EFUSE_OUT5                     | L99SP08_CH2               |                                                                                                                                                                                                                  |
|           | EFUSE_OUT6                     | L99SP08_CH1               |                                                                                                                                                                                                                  |
|           | EFUSE_OUT7                     | L99SP08_CH0               |                                                                                                                                                                                                                  |
| VNF9D1M5Q | Channel 0                      | DEV0, VNF9D1M5Q_CHANNEL_1 | These mapping is used both in the EFUSE set function and in the EEPROM index layout.                                                                                                                             |
|           | Channel 1                      | DEV0, VNF9D1M5Q_CHANNEL_0 |                                                                                                                                                                                                                  |
|           | Channel 2                      | DEV1, VNF9D1M5Q_CHANNEL_1 |                                                                                                                                                                                                                  |
|           | Channel 3                      | DEV1, VNF9D1M5Q_CHANNEL_0 |                                                                                                                                                                                                                  |
| VNF9D5SF  | Channel 4                      | VNF9D5SF_DEV0_CHANNEL_1   | UDS requests including other channels are rejected.                                                                                                                                                              |
|           | Channel 5                      | VNF9D5SF_DEV0_CHANNEL_0   |                                                                                                                                                                                                                  |
| VNF9Q20SF | Channel 6                      | VNF9Q20SF_DEV0_CHANNEL_2  | These mapping is used both in the EFUSE logic and in the EEPROM indexes.                                                                                                                                         |
|           | Channel 7                      | VNF9Q20SF_DEV0_CHANNEL_1  |                                                                                                                                                                                                                  |

| Device    | AEK-POW-PDUMINI output channel | Device channel           | Notes                                                                    |
|-----------|--------------------------------|--------------------------|--------------------------------------------------------------------------|
| VNF9Q20SF | Channel 8                      | VNF9Q20SF_DEV0_CHANNEL_0 | These mapping is used both in the EFUSE logic and in the EEPROM indexes. |
|           | Channel 9                      | VNF9Q20SF_DEV0_CHANNEL_3 |                                                                          |

#### A.4.2 EEPROM layout and handling

All configurable thresholds and I<sup>2</sup>t parameters are collected in a single, static, structured block in the EEPROM. This block is identified by a logical EEPROM ID. At boot, this block is read, validated and used to reconstruct the RAM configuration and to reprogram the hardware registers. If any parameter is changed via UDS, only the relevant bytes in the RAM array are updated, and the entire block is rewritten into the EEPROM.

All thresholds and I<sup>2</sup>t parameters are kept in a single configuration array, stored in the EEPROM and mirrored in the RAM.

If the validity signature is wrong or missing, the system safely reverts to known defaults and rewrites a consistent configuration block.

#### A.5 Device configuration functions

The EFUSE service layer exposes a set of device configuration functions that abstract all low-level register accesses and EEPROM handling using simple, channel-oriented APIs. Each function is responsible for validating the input parameters, mapping the logical PDU channel to the appropriate device and hardware channel, programming the device registers, as well as optionally performing a read-back to check any change, and updating the persistent configuration block in the EEPROM.

This ensures that every change requested via UDS is applied consistently to the hardware and stored in the non-volatile memory to ensure the same configuration is loaded at the next power-on.

For each device family, the following subsections describe which function is provided, the related parameters, and the function interaction with the underlying drivers and the EEPROM layout.

#### A.6 L99SP08 threshold settings

Steps:

1. Validate the threshold (0..3).
2. Map channel to the corresponding *L99SP08\_CHx*.
3. Call *L99SP08\_ch\_setIThreshold()* to program the register.
4. Call *L99SP08\_ch\_getITh()* to read back and verify.
5. If successful, update EEPROM via *EFUSE\_sev\_saveChannellthrToEEPROM(...)*.

#### A.7 VNF9D1M5Q I<sup>2</sup>t settings

Steps:

1. Validate:
  - ``inom` ∈ [0..7]` (``VNF9D1M5Q_inom_level_f``).
  - ``tnom` ∈ [0..7]` (``VNF9D1M5Q_tnom_level_f``).
  - ``ipeak` ∈ [0..1]` (``VNF9D1M5Q_ipeak_level_f``).
2. Map AEK-POW-PDUMINI channel (0..3) to ``dev`` and ``VNF9D1M5Q_ch_f``.
3. Call *VNF9D1M5Q\_ch\_setI2TConf(dev, ch, inom\_level, ipeak\_level, tnom\_level)*.
4. Read back using ``VNF9D1M5Q_ch_getInom/Tnom/Ipeak``.
5. If successful, update EEPROM via ``EFUSE_sev_saveVNF9D1M5Q_I2T_ToEEPROM(...)``.

#### A.8 VNF9D5SF I<sup>2</sup>t settings

Only ``channel` = EFUSE_OUT4` or ``EFUSE_OUT5`` are accepted.



The steps to follow are like the other devices: validate, map, set I<sup>2</sup>t via `VNF9D5SF_ch_setI2TConf`, then save to EEPROM (`EFUSE_sev_saveVNF9D5SF_I2T_ToEEPROM`).

## A.9 VNF9Q20SF I<sup>2</sup>t settings

Valid channels are `EFUSE_OUT6..EFUSE_OUT9`.

Validation, mapping and EEPROM handling are analogous.

## Appendix B UDS usage examples

This section describes how to switch from Standard Diagnostic Session to Extended Diagnostic Session. It also describes how to set the **L99SP08** current threshold ( $I_{THR}$ ) for a specific **AEK-POW-PDUMINI** channel using UDS.

### B.1 Switch from Standard Session to Extended Session

To enable configuration services (e.g. threshold programming), switch the ECU from the default Standard Diagnostic Session to Extended Diagnostic Session using the *DiagnosticSessionControl* (SID 0x10) UDS service.

#### UDS Request Format

Request (CAN ID = 0x7E0)

**Table 15.** Extended diagnostic session details

| Byte | Value | Description                              |
|------|-------|------------------------------------------|
| 0    | 0x02  | Length (2 bytes of UDS data)             |
| 1    | 0x10  | SID = DiagnosticSessionControl           |
| 2    | 0x03  | Sub-function = ExtendedDiagnosticSession |
| 3–7  | 0x00  | Padding                                  |

**Figure 164.** Extended diagnostic session: data byte view

| Data Byte View (HEX) |    |    |    |    |    |    |    |    |
|----------------------|----|----|----|----|----|----|----|----|
| Index                | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 000                  | 02 | 10 | 03 | 00 | 00 | 00 | 00 | 00 |

Flow:

- The tester requests to enter Extended Diagnostic Session (0x03).
- The ECU will answer with a positive response (0x50 0x03 ...) if the session switch is accepted.

### B.2 Set L99SP08 $I_{THR}$ threshold using UDS

Once the ECU enters Extended Diagnostic Session, set the **L99SP08** current threshold for an **AEK-POW-PDUMINI** channel using *WriteDataByIdentifier* (SID 0x2E) and the *DID CHANNEL\_ITHR\_L99SP08* (0xF12A).

#### UDS Request Format

Request (CAN ID = 0x7E0)

**Table 16.** Current threshold settings for L99SP08

| Byte | Value | Description                                     |
|------|-------|-------------------------------------------------|
| 0    | 0x05  | Length (4 bytes of UDS data)                    |
| 1    | 0x2E  | SID = WriteDataByIdentifier                     |
| 2    | 0xF1  | DID High                                        |
| 3    | 0x2A  | DID Low → CHANNEL_ITHR_L99SP08 (0xF12A)         |
| 4    | 0x00  | Channel ( <b>AEK-POW-PDUMINI</b> channel index) |
| 5    | 0x03  | Threshold code ( $I_{THR}$ level)               |
| 6–7  | 0x00  | Padding                                         |

**Figure 165. Current threshold settings for L99SP08: data byte view**

| Data Byte View (HEX) |    |    |    |    |    |    |    |    |
|----------------------|----|----|----|----|----|----|----|----|
| Index                | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 000                  | 05 | 2E | F1 | 2A | 00 | 03 | 00 | 00 |

**Table 17. Current threshold settings for L99SP08: field description**

| Field  | Byte(s) | Description                           |
|--------|---------|---------------------------------------|
| LEN    | 0       | Number of UDS bytes (here 5)          |
| SID    | 1       | 0x2E – WriteDataByIdentifier          |
| DID    | 2–3     | 0xF12A – L99SP08 I_THR threshold      |
| CHAN   | 4       | AEK-POW-PDUMINI channel (EFUSE\_OUTx) |
| THRESH | 5       | Encoded threshold value (0..3)        |
| PAD    | 6–7     | Unused / padding bytes                |

### B.3 Channel mapping for L99SP08

Only some AEK-POW-PDUMINI channels are mapped to the L99SP08 device as detailed in the table below.

**Table 18. L99SP08 channel mapping**

| AEK-POW-PDUMINI channel (CHAN) | Mapping                                 |
|--------------------------------|-----------------------------------------|
| 0x00                           | AEK-POW-PDUMINI channel 0 → L99SP08 CH7 |
| 0x01                           | AEK-POW-PDUMINI channel 1 → L99SP08 CH6 |
| 0x02                           | AEK-POW-PDUMINI channel 2 → L99SP08 CH5 |
| 0x03                           | AEK-POW-PDUMINI channel 3 → L99SP08 CH4 |
| 0x04                           | AEK-POW-PDUMINI channel 4 → L99SP08 CH3 |
| 0x05                           | AEK-POW-PDUMINI channel 5 → L99SP08 CH2 |
| 0x06                           | AEK-POW-PDUMINI channel 6 → L99SP08 CH1 |
| 0x07                           | AEK-POW-PDUMINI channel 7 → L99SP08 CH0 |

If a UDS request uses a channel not supported by L99SP08 (e.g. 8 or 9), the ECU returns a negative response (*NRC RequestOutOfRange*).

#### Threshold Encoding (I\_THR)

The threshold byte (THRESH, Byte 5) encodes the desired current threshold according to the L99SP08 driver enumeration.

**Table 19. Threshold bytes and related L99SP08 current threshold**

| THRESH value | L99SP08 I_THR level | Nominal current |
|--------------|---------------------|-----------------|
| 0x00         | L99SP08_ITHR100MA   | 100 mA          |
| 0x01         | L99SP08_ITHR200MA   | 200 mA          |
| 0x02         | L99SP08_ITHR300MA   | 300 mA          |
| 0x03         | L99SP08_ITHR400MA   | 400 mA          |
| 0x04[...0xFF | –                   | Not supported   |

If the tester sends THRESH > 0x03, the EFUSE logic rejects the value and the UDS response is a negative response (*RequestOutOfRange*).

## B.4 Setting I<sup>2</sup>T VNF9D1M5Q thresholds using UDS

This section describes how to configure the **VNF9D1M5Q** I<sup>2</sup>T thresholds ( $I_{NOM}$ ,  $T_{NOM}$ ,  $I_{PEAK}$ ) via UDS using *WriteDataByIdentifier* (SID 0x2E) and the dedicated DID.

### UDS Request Format

**CAN ID:** 0x7E0 (UDS request ID)

**Service:** 0x2E – WriteDataByIdentifier

**DID:** CHANNEL\_I2T\_VNFD1M5Q = 0xF12B (DID High 0xF1, DID Low 0x2B)

### Example request frame

**Table 20. Current threshold settings for VNF9D1M5Q**

| Byte | Value | Description                                       |
|------|-------|---------------------------------------------------|
| 0    | 0x07  | Length (7 bytes of UDS data)                      |
| 1    | 0x2E  | SID = `WriteDataByIdentifier`                     |
| 2    | 0xF1  | DID High                                          |
| 3    | 0x2B  | DID Low → `CHANNEL_I2T_VNFD1M5Q`                  |
| 4    | 0x02  | CHAN (PDU channel index)                          |
| 5    | 0x07  | $I_{NOM}$ (encoded nominal current)               |
| 6    | 0x07  | $T_{NOM}$ (encoded nominal time)                  |
| 7    | 0x00  | $I_{PEAK}$ (encoded I <sup>2</sup> T peak factor) |

**Figure 166. Current threshold settings for VNF9D1M5Q: data byte view**

| Data Byte View (HEX) |    |    |    |    |    |    |    |    |
|----------------------|----|----|----|----|----|----|----|----|
| Index                | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 000                  | 07 | 2E | F1 | 2B | 02 | 07 | 07 | 00 |

### Field meaning

**Table 21. Current threshold settings for VNF9D1M5Q : field description**

| Field    | Byte(s) | Description                       |
|----------|---------|-----------------------------------|
| LEN      | 0       | Number of UDS data bytes (here 6) |
| SID      | 1       | 0x2E – WriteDataByIdentifier      |
| DID High | 2       | 0xF1                              |
| DID Low  | 3       | 0x2B → DID CHANNEL_I2T_VNFD1M5Q   |
| CHAN     | 4       | PDU channel (EFUSE_sev_channel_t) |
| INOM     | 5       | Encoded nominal current level     |
| TNOM     | 6       | Encoded nominal time level        |
| IPEAK    | 7       | Encoded peak factor level         |

### Channel support for VNF9D1M5Q

Only **AEK-POW-PDUMINI** channels 0, 1, 2, 3 are mapped to the **VNF9D1M5Q**.

**Table 22. AEK-POW-PDUMINI and VNF9D1M5Q channel mapping**

| AEK-POW-PDUMINI channel | Device mapping            |
|-------------------------|---------------------------|
| PDU ch0                 | VNF9D1M5Q DEV0, CHANNEL_1 |

| AEK-POW-PDUMINI channel | Device mapping            |
|-------------------------|---------------------------|
| PDU ch1                 | VNF9D1M5Q DEV0, CHANNEL_0 |
| PDU ch2                 | VNF9D1M5Q DEV1, CHANNEL_1 |
| PDU ch3                 | VNF9D1M5Q DEV1, CHANNEL_0 |

If a UDS request uses a channel outside 0..3 for DID 0xF12B, the ECU responds with negative response (*RequestOutOfRange*).

#### **I<sub>NOM</sub> (nominal current) encoding**

The nominal current parameter of the **VNF9D1M5Q** is encoded as a numeric value in the diagnostic message. This encoding allows the diagnostic request to carry a compact coded value, which the ECU interprets as the corresponding physical current threshold.

In fact, each code physically corresponds to setting the nominal current threshold to a specific value.

The supported encodings are:

- **VNF9D1M5Q\_INOM\_8A**: physically sets the nominal current threshold to 8 A. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_9A**: physically sets the nominal current threshold to 9 A. To select this setting, value 1 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_12A**: physically sets the nominal current threshold to 12 A. To select this setting, value 2 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_14A**: physically sets the nominal current threshold to 14 A. To select this setting, value 3 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_16A**: physically sets the nominal current threshold to 16 A. To select this setting, value 4 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_21A**: physically sets the nominal current threshold to 21 A. To select this setting, value 5 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_23A**: physically sets the nominal current threshold to 23 A. To select this setting, value 6 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_INOM\_27A**: physically sets the nominal current threshold to 27 A. To select this setting, value 7 shall be transmitted in the CAN message.

**Table 23. I<sub>NOM</sub> value table**

| I <sub>NOM</sub> byte | Enum               | Nominal current |
|-----------------------|--------------------|-----------------|
| 0x00                  | VNF9D1M5Q_INOM_8A  | 8 A             |
| 0x01                  | VNF9D1M5Q_INOM_9A  | 9 A             |
| 0x02                  | VNF9D1M5Q_INOM_12A | 12 A            |
| 0x03                  | VNF9D1M5Q_INOM_14A | 14 A            |
| 0x04                  | VNF9D1M5Q_INOM_16A | 16 A            |
| 0x05                  | VNF9D1M5Q_INOM_21A | 21 A            |
| 0x06                  | VNF9D1M5Q_INOM_23A | 23 A            |
| 0x07                  | VNF9D1M5Q_INOM_27A | 27 A            |
| 0x08[...].FF          | –                  | Not supported   |

Values above 0x07 are rejected (*RequestOutOfRange*).

#### **T<sub>NOM</sub> (nominal time) encoding**

The nominal time parameter of the **VNF9D1M5Q** is encoded as a numeric value in the diagnostic message.

This encoding allows the diagnostic request to carry a compact coded value, which the ECU interprets as the corresponding physical nominal time. In fact, each code physically corresponds to setting the nominal time threshold to a specific value.

The supported encodings are:



- **VNF9D1M5Q\_TNOM\_300S**: physically sets the nominal time threshold to 300 s. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_257S**: physically sets the nominal time threshold to 257 s. To select this setting, value 1 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_214S**: physically sets the nominal time threshold to 214 s. To select this setting, value 2 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_172S**: physically sets the nominal time threshold to 172 s. To select this setting, value 3 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_129S**: physically sets the nominal time threshold to 129 s. To select this setting, value 4 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_86S**: physically sets the nominal time threshold to 86 s. To select this setting, value 5 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_44S**: physically sets the nominal time threshold to 44 s. To select this setting, value 6 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_TNOM\_1S**: physically sets the nominal time threshold to 1 s. To select this setting, value 7 shall be transmitted in the CAN message.

**Table 24. T<sub>NOM</sub> value table**

| T <sub>NOM</sub> byte | Enum                | Nominal time  |
|-----------------------|---------------------|---------------|
| 0x00                  | VNF9D1M5Q_TNOM_300S | 300 s         |
| 0x01                  | VNF9D1M5Q_TNOM_257S | 257 s         |
| 0x02                  | VNF9D1M5Q_TNOM_214S | 214 s         |
| 0x03                  | VNF9D1M5Q_TNOM_172S | 172 s         |
| 0x04                  | VNF9D1M5Q_TNOM_129S | 129 s         |
| 0x05                  | VNF9D1M5Q_TNOM_86S  | 86 s          |
| 0x06                  | VNF9D1M5Q_TNOM_44S  | 44 s          |
| 0x07                  | VNF9D1M5Q_TNOM_1S   | 1 s           |
| 0x08[...].JFF         | –                   | Not supported |

T<sub>NOM</sub> values > 0x07 are invalid.

#### **I<sub>PEAK</sub> (peak factor) encoding**

The peak current parameter of the **VNF9D1M5Q** is encoded as a numeric value in the diagnostic message.

This encoding allows the diagnostic request to carry a compact coded value, which the ECU interprets as the corresponding physical peak current. In fact, each code physically corresponds to setting the peak current threshold as a multiple of the nominal current.

The supported encodings are:

- **VNF9D1M5Q\_IPEAK\_10X\_INOM**: physically sets the peak current threshold to 10 times the nominal current. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9D1M5Q\_IPEAK\_6X\_INOM**: physically sets the peak current threshold to 6 times the nominal current. To select this setting, value 1 shall be transmitted in the CAN message.

**Table 25. I<sub>PEAK</sub> value table**

| I <sub>PEAK</sub> byte | Enum                     | Description      |
|------------------------|--------------------------|------------------|
| 0x00                   | VNF9D1M5Q_IPEAK_10X_INOM | Peak = 10 × INOM |
| 0x01                   | VNF9D1M5Q_IPEAK_6X_INOM  | Peak = 6 × INOM  |
| 0x02[...].JFF          | –                        | Not supported    |

Values > 0x01 are rejected by the EFUSE logic.

## B.5 Setting VNF9D5SF I<sup>2</sup>t thresholds using UDS

This section describes how to configure the **VNF9D5SF** I<sup>2</sup>t thresholds via UDS using *WriteDataByIdentifier* (SID 0x2E) and the dedicated DID.

### UDS Request Format

**CAN ID:** 0x7E0 (UDS request ID)

**Service:** 0x2E – WriteDataByIdentifier

**DID:** CHANNEL\_I2T\_VNF9D5SF = 0xF12C (DID High 0xF1, DID Low 0x2C)

**Table 26. Example request frame**

| Byte | Value | Description                                   |
|------|-------|-----------------------------------------------|
| 0    | 0x06  | Length (6 bytes of UDS data)                  |
| 1    | 0x2E  | SID = WriteDataByIdentifier                   |
| 2    | 0xF1  | DID High                                      |
| 3    | 0x2C  | DID Low → CHANNEL_I2T_VNF9D5SF                |
| 4    | 0x05  | CHAN (AEK-POW-PDUMINI channel index)          |
| 5    | 0x07  | I <sub>NOM</sub> (encoded nominal current)    |
| 6    | 0x07  | T <sub>NOM</sub> (encoded nominal time)       |
| 7    | 0x07  | I <sub>SHUTOFF</sub> (encoded shutoff factor) |

**Figure 167. Example request frame: data byte view**

| Data Byte View (HEX) |    |    |    |    |    |    |    |    |
|----------------------|----|----|----|----|----|----|----|----|
| Index                | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 000                  | 06 | 2E | F1 | 2C | 05 | 07 | 07 | 07 |

**Table 27. Field meaning**

| Field    | Byte(s) | Description                                   |
|----------|---------|-----------------------------------------------|
| LEN      | 0       | Number of UDS data bytes (here 6)             |
| SID      | 1       | 0x2E – WriteDataByIdentifier                  |
| DID High | 2       | 0xF1                                          |
| DID Low  | 3       | 0x2C → DID CHANNEL_I2T_VNF9D5SF               |
| CHAN     | 4       | AEK-POW-PDUMINI channel (EFUSE_sev_channel_t) |
| INOM     | 5       | Encoded nominal current level                 |
| TNOM     | 6       | Encoded nominal time level                    |
| ISHUTOFF | 7       | Encoded shutoff threshold level               |

### Channel support for VNF9D5SF

Only **AEK-POW-PDUMINI** channels 4 and 5 are mapped to the VNF9D5SF.

**Table 28. AEK-POW-PDUMINI and VNF9D5SF channel mapping**

| Channel value | AEK-POW-PDUMINI channel | Mapping                  |
|---------------|-------------------------|--------------------------|
| 0x04          | PDU ch4                 | VNF9D5SF DEV0, CHANNEL 1 |
| 0x05          | PDU ch5                 | VNF9D5SF DEV0, CHANNEL 0 |

If a UDS request uses a channel different from 0x04 or 0x05 for DID 0xF12C, the ECU responds with a negative response (*RequestOutOfRange*).

#### **I<sub>NOM</sub> (nominal current) encoding**

The nominal current parameter of the **VNF9D5SF** is encoded as a numeric value in the diagnostic message.

Each code physically corresponds to setting the nominal current threshold to a specific value.

The supported encodings are:

- **VNF9D5SF\_CH\_INOM\_15A**: physically sets the nominal current threshold to 15 A. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_4A**: physically sets the nominal current threshold to 4 A. To select this setting, value 1 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_6A**: physically sets the nominal current threshold to 6 A. To select this setting, value 2 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_7\_5A**: physically sets the nominal current threshold to 7.5 A. To select this setting, value 3 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_9**: physically sets the nominal current threshold to 9 A. To select this setting, value 4 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_10A**: physically sets the nominal current threshold to 10 A. To select this setting, value 5 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_11\_5A**: physically sets the nominal current threshold to 11.5 A. To select this setting, value 6 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_INOM\_13A**: physically sets the nominal current threshold to 13 A. To select this setting, value 7 shall be transmitted in the CAN message.

**Table 29. I<sub>NOM</sub> value table**

| I <sub>NOM</sub> byte | Enum                   | Nominal current |
|-----------------------|------------------------|-----------------|
| 0x00                  | VNF9D5SF_CH_INOM_15A   | 15 A            |
| 0x01                  | VNF9D5SF_CH_INOM_4A    | 4 A             |
| 0x02                  | VNF9D5SF_CH_INOM_6A    | 6 A             |
| 0x03                  | VNF9D5SF_CH_INOM_7_5A  | 7.5 A           |
| 0x04                  | VNF9D5SF_CH_INOM_9     | 9 A             |
| 0x05                  | VNF9D5SF_CH_INOM_10A   | 10 A            |
| 0x06                  | VNF9D5SF_CH_INOM_11_5A | 11.5 A          |
| 0x07                  | VNF9D5SF_CH_INOM_13A   | 13 A            |
| 0x08[...] <b>FF</b>   | –                      | Not supported   |

Any I<sub>NOM</sub> value greater than 0x07 is rejected by EFUSE logic as out of range.

#### **T<sub>NOM</sub> (nominal time) encoding**

The T<sub>NOM</sub> parameter encoding is the same as the one already defined for the VNF9D1M5Q. The same encoding also applies to VNF9D5SF and VNF9Q20SF.

For this reason, the encoding values are not repeated in this section and shall be considered as described above.

**Table 30. T<sub>NOM</sub> value table**

| T <sub>NOM</sub> byte | Enum                  | Nominal time |
|-----------------------|-----------------------|--------------|
| 0x00                  | VNF9D5SF_CH_TNOM_300S | 300 s        |
| 0x01                  | VNF9D5SF_CH_TNOM_257S | 257 s        |
| 0x02                  | VNF9D5SF_CH_TNOM_214S | 214 s        |
| 0x03                  | VNF9D5SF_CH_TNOM_172S | 172 s        |

| T <sub>NOM</sub> byte | Enum                  | Nominal time  |
|-----------------------|-----------------------|---------------|
| 0x04                  | VNF9D5SF_CH_TNOM_129S | 129 s         |
| 0x05                  | VNF9D5SF_CH_TNOM_86S  | 86 s          |
| 0x06                  | VNF9D5SF_CH_TNOM_44S  | 44 s          |
| 0x07                  | VNF9D5SF_CH_TNOM_1S   | 1 s           |
| 0x08[...].FF          | –                     | Not supported |

T<sub>NOM</sub> values greater than 0x07 are invalid.

#### **ISHUTOFF (shutoff current factor) encoding**

The shutoff current parameter of the **VNF9D5SF** is encoded as a numeric value in the diagnostic message.

Each code physically corresponds to setting the shutoff current threshold as a multiple of the nominal current, or to disabling the shutoff function.

The supported encodings are:

- **VNF9D5SF\_CH\_ISHUT\_OFF**: physically disables the shutoff current threshold. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_2X**: physically sets the shutoff current threshold to 2 times the nominal current. To select this setting, value 1 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_3X**: physically sets the shutoff current threshold to 3 times the nominal current. To select this setting, value 2 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_3\_6X**: physically sets the shutoff current threshold to 3.6 times the nominal current. To select this setting, value 3 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_4\_1X**: physically sets the shutoff current threshold to 4.1 times the nominal current. To select this setting, value 4 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_5X**: physically sets the shutoff current threshold to 5 times the nominal current. To select this setting, value 5 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_5\_7X**: physically sets the shutoff current threshold to 5.7 times the nominal current. To select this setting, value 6 shall be transmitted in the CAN message.
- **VNF9D5SF\_CH\_ISHUT\_6\_5X**: physically sets the shutoff current threshold to 6.5 times the nominal current. To select this setting, value 7 shall be transmitted in the CAN message.

**Table 31. ISHUTOFF value table**

| ISHUTOFF byte | Enum                   | Description          |
|---------------|------------------------|----------------------|
| 0x00          | VNF9D5SF_CH_ISHUT_OFF  | Shutoff disabled     |
| 0x01          | VNF9D5SF_CH_ISHUT_2X   | 2× nominal current   |
| 0x02          | VNF9D5SF_CH_ISHUT_3X   | 3× nominal current   |
| 0x03          | VNF9D5SF_CH_ISHUT_3_6X | 3.6× nominal current |
| 0x04          | VNF9D5SF_CH_ISHUT_4_1X | 4.1× nominal current |
| 0x05          | VNF9D5SF_CH_ISHUT_5X   | 5× nominal current   |
| 0x06          | VNF9D5SF_CH_ISHUT_5_7X | 5.7× nominal current |
| 0x07          | VNF9D5SF_CH_ISHUT_6_5X | 6.5× nominal current |
| 0x08[...].FF  | –                      | Not supported        |

## **B.6 Setting VNF9Q20SF I<sub>t</sub> thresholds using UDS**

This section describes how to configure the **VNF9Q20SF** I<sub>t</sub> thresholds (I<sub>NOM</sub>, T<sub>NOM</sub>, I<sub>SHUTOFF</sub>) via UDS using *WriteDataByIdentifier* (SID 0x2E) and the dedicated DID.

#### **UDS request format**

**CAN ID:** 0x7E0 (UDS request ID)

**Service:** 0x2E – WriteDataByIdentifier

**DID:** CHANNEL\_I2T\_VNF9Q20SF = 0xF12D (DID High 0xF1, DID Low 0x2D)

**Table 32. Example request frame**

| Byte | Value | Description                                   |
|------|-------|-----------------------------------------------|
| 0    | 0x06  | Length (6 bytes of UDS data)                  |
| 1    | 0x2E  | SID = WriteDataByIdentifier                   |
| 2    | 0xF1  | DID High                                      |
| 3    | 0x2D  | DID Low → CHANNEL_I2T_VNF9Q20SF               |
| 4    | 0x09  | CHAN (AEK-POW-PDUMINI channel index)          |
| 5    | 0x07  | I <sub>NOM</sub> (encoded nominal current)    |
| 6    | 0x07  | T <sub>NOM</sub> (encoded nominal time)       |
| 7    | 0x07  | I <sub>SHUTOFF</sub> (encoded shutoff factor) |

**Figure 168. Example request frame: data byte view**

| Data Byte View (HEX) |    |    |    |    |    |    |    |    |
|----------------------|----|----|----|----|----|----|----|----|
| Index                | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
| 000                  | 06 | 2E | F1 | 2D | 09 | 07 | 07 | 07 |

**Table 33. Field meaning**

| Field    | Byte(s) | Description                                   |
|----------|---------|-----------------------------------------------|
| LEN      | 0       | Number of UDS data bytes (here 6)             |
| SID      | 1       | 0x2E – WriteDataByIdentifier                  |
| DID High | 2       | 0xF1                                          |
| DID Low  | 3       | 0x2D → DID CHANNEL_I2T_VNF9Q20SF              |
| CHAN     | 4       | AEK-POW-PDUMINI channel (EFUSE_sev_channel_t) |
| INOM     | 5       | Encoded nominal current level                 |
| TNOM     | 6       | Encoded nominal time level                    |
| ISHUTOFF | 7       | Encoded shutoff factor level                  |

#### Channel support for VNF9Q20SF

Only **AEK-POW-PDUMINI** channels 6, 7, 8, 9 are mapped to the **VNF9Q20SF**.

**Table 34. AEK-POW-PDUMINI and VNF9Q20SF channel mapping**

| CHAN | AEK-POW-PDUMINI channel | Device mapping            |
|------|-------------------------|---------------------------|
| 0x06 | PDU ch6                 | VNF9Q20SF DEV0, CHANNEL_2 |
| 0x07 | PDU ch7                 | VNF9Q20SF DEV0, CHANNEL_1 |
| 0x08 | PDU ch8                 | VNF9Q20SF DEV0, CHANNEL_0 |
| 0x09 | PDU ch9                 | VNF9Q20SF DEV0, CHANNEL_3 |

If a UDS request uses a channel outside 6..9 for DID 0xF12D, the ECU responds with negative response (*RequestOutOfRange*).

#### I<sub>NOM</sub> (nominal current) encoding

The nominal current parameter of the **VNF9Q20SF** is encoded as a numeric value in the diagnostic message.

Each code physically corresponds to setting the nominal current threshold to a specific value.

The supported encodings are:

- **VNF9Q20SF\_CH\_INOM\_6A**: physically sets the nominal current threshold to 6 A. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_1\_5A**: physically sets the nominal current threshold to 1.5 A. To select this setting, value 1 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_2A**: physically sets the nominal current threshold to 2 A. To select this setting, value 2 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_2\_5A**: physically sets the nominal current threshold to 2.5 A. To select this setting, value 3 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_3A**: physically sets the nominal current threshold to 3 A. To select this setting, value 4 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_3\_5A**: physically sets the nominal current threshold to 3.5 A. To select this setting, value 5 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_4A**: physically sets the nominal current threshold to 4 A. To select this setting, value 6 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_INOM\_5A**: physically sets the nominal current threshold to 5 A. To select this setting, value 7 shall be transmitted in the CAN message.

**Table 35. I<sub>NOM</sub> value table**

| I <sub>NOM</sub> byte | Enum                   | Nominal current |
|-----------------------|------------------------|-----------------|
| 0x00                  | VNF9Q20SF_CH_INOM_6A   | 6 A             |
| 0x01                  | VNF9Q20SF_CH_INOM_1_5A | 1.5 A           |
| 0x02                  | VNF9Q20SF_CH_INOM_2A   | 2 A             |
| 0x03                  | VNF9Q20SF_CH_INOM_2_5A | 2.5 A           |
| 0x04                  | VNF9Q20SF_CH_INOM_3A   | 3 A             |
| 0x05                  | VNF9Q20SF_CH_INOM_3_5A | 3.5 A           |
| 0x06                  | VNF9Q20SF_CH_INOM_4A   | 4 A             |
| 0x07                  | VNF9Q20SF_CH_INOM_5A   | 5 A             |
| 0x08[...].FF          | –                      | Not supported   |

Values above 0x07 are rejected as out of range.

#### **T<sub>NOM</sub> (nominal time) encoding**

For **VNF9Q20SF**, T<sub>NOM</sub> encoding is the same as the one already defined for the VNF9D1M5Q device. The same encoding also applies to **VNF9D5SF** and **VNF9Q20SF** devices.

For this reason, the encoding values are not repeated in this section and shall be considered as described above.

**Table 36. T<sub>NOM</sub> value table**

| T <sub>NOM</sub> byte | Enum                   | Nominal time |
|-----------------------|------------------------|--------------|
| 0x00                  | VNF9Q20SF_CH_TNOM_300S | 300 s        |
| 0x01                  | VNF9Q20SF_CH_TNOM_257S | 257 s        |
| 0x02                  | VNF9Q20SF_CH_TNOM_214S | 214 s        |
| 0x03                  | VNF9Q20SF_CH_TNOM_172S | 172 s        |
| 0x04                  | VNF9Q20SF_CH_TNOM_129S | 129 s        |
| 0x05                  | VNF9Q20SF_CH_TNOM_86S  | 86 s         |
| 0x06                  | VNF9Q20SF_CH_TNOM_44S  | 44 s         |
| 0x07                  | VNF9Q20SF_CH_TNOM_1S   | 1 s          |



| T <sub>NOM</sub> byte | Enum | Nominal time  |
|-----------------------|------|---------------|
| 0x08[...]FF           | –    | Not supported |

#### ISHUTOFF (shutoff current factor) encoding

The supported encodings are:

- **VNF9Q20SF\_CH\_ISHUT\_OFF**: physically disables the shutoff current threshold. To select this setting, value 0 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_2X**: physically sets the shutoff current threshold to 2 times the nominal current. To select this setting, value 1 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_3X**: physically sets the shutoff current threshold to 3 times the nominal current. To select this setting, value 2 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_3\_6X**: physically sets the shutoff current threshold to 3.6 times the nominal current. To select this setting, value 3 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_4\_1X**: physically sets the shutoff current threshold to 4.1 times the nominal current. To select this setting, value 4 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_5X**: physically sets the shutoff current threshold to 5 times the nominal current. To select this setting, value 5 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_5\_7X**: physically sets the shutoff current threshold to 5.7 times the nominal current. To select this setting, value 6 shall be transmitted in the CAN message.
- **VNF9Q20SF\_CH\_ISHUT\_6\_5X**: physically sets the shutoff current threshold to 6.5 times the nominal current. To select this setting, value 7 shall be transmitted in the CAN message.

**Table 37. ISHUTOFF value table**

| ISHUTOFF byte | Enum                    | Description          |
|---------------|-------------------------|----------------------|
| 0x00          | VNF9Q20SF_CH_ISHUT_OFF  | Shutoff disabled     |
| 0x01          | VNF9Q20SF_CH_ISHUT_2X   | 2× nominal current   |
| 0x02          | VNF9Q20SF_CH_ISHUT_3X   | 3× nominal current   |
| 0x03          | VNF9Q20SF_CH_ISHUT_3_6X | 3.6× nominal current |
| 0x04          | VNF9Q20SF_CH_ISHUT_4_1X | 4.1× nominal current |
| 0x05          | VNF9Q20SF_CH_ISHUT_5X   | 5× nominal current   |
| 0x06          | VNF9Q20SF_CH_ISHUT_5_7X | 5.7× nominal current |
| 0x07          | VNF9Q20SF_CH_ISHUT_6_5X | 6.5× nominal current |
| 0x08[...]FF   | –                       | Not supported        |

## Revision history

**Table 38. Document revision history**

| Date        | Revision | Changes                                                                                                                         |
|-------------|----------|---------------------------------------------------------------------------------------------------------------------------------|
| 26-Feb-2026 | 1        | Initial release.                                                                                                                |
| 26-May-2026 | 2        | Added Section 6.2: Additional demos, Appendix A: UDS extensions and threshold configuration and Appendix B: UDS usage examples. |

## Contents

|          |                                                                       |           |
|----------|-----------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Overview</b>                                                       | <b>2</b>  |
| 1.1      | Deliverable package contents                                          | 2         |
| 1.2      | Supported key devices                                                 | 2         |
| 1.3      | Technology background on high-side drivers with STi <sup>2</sup> Fuse | 3         |
| 1.4      | Reverse polarity                                                      | 4         |
| 1.5      | Shortcut to ground                                                    | 4         |
| 1.6      | ISO pulses                                                            | 4         |
| 1.7      | ESD                                                                   | 4         |
| <b>2</b> | <b>Hardware description</b>                                           | <b>5</b>  |
| 2.1      | Hardware features                                                     | 5         |
| 2.2      | Hardware dimensions                                                   | 7         |
| 2.3      | Main connector                                                        | 7         |
| 2.4      | Power input connectors                                                | 9         |
| 2.5      | Power output connectors                                               | 10        |
| 2.6      | CAN communication connector                                           | 11        |
| 2.7      | Assembly mode                                                         | 11        |
| 2.8      | Parking mode                                                          | 11        |
| 2.9      | Status LEDs for debugging                                             | 12        |
| 2.10     | L99SP08 block diagram                                                 | 13        |
| 2.11     | STi <sup>2</sup> Fuse devices                                         | 14        |
| 2.11.1   | VNF9D1M5Q block diagram                                               | 15        |
| 2.11.2   | VNF9D5SF block diagram                                                | 16        |
| 2.11.3   | VNF9Q20SF block diagram                                               | 18        |
| 2.12     | SPSB081 block diagram                                                 | 19        |
| 2.13     | VN7140AJ block diagram                                                | 20        |
| <b>3</b> | <b>Hardware test results</b>                                          | <b>22</b> |
| 3.1      | Test setup                                                            | 22        |
| 3.2      | Used tools                                                            | 22        |
| 3.3      | Sleep, wake-up, startup time conditions                               | 22        |
| 3.3.1    | Current consumption                                                   | 22        |
| 3.3.2    | Wake-up sources and startup times                                     | 22        |
| 3.4      | Regular I <sup>2</sup> t functionality                                | 27        |
| 3.4.1    | Test conditions                                                       | 27        |
| 3.4.2    | VNF9D1M5Q                                                             | 28        |

|            |                                                             |           |
|------------|-------------------------------------------------------------|-----------|
| 3.4.3      | VNF9D5SF .....                                              | 34        |
| 3.4.4      | VNF9Q20SF .....                                             | 37        |
| 3.4.5      | Results' summary .....                                      | 43        |
| <b>3.5</b> | <b>Maximum current on all channels .....</b>                | <b>43</b> |
| 3.5.1      | Test conditions .....                                       | 43        |
| 3.5.2      | Individual channels .....                                   | 43        |
| 3.5.3      | Combination of devices for maximum current .....            | 50        |
| 3.5.4      | Results discussion .....                                    | 52        |
| <b>3.6</b> | <b>Independent operation of multichannel devices .....</b>  | <b>52</b> |
| 3.6.1      | Test conditions .....                                       | 52        |
| 3.6.2      | VNF9D1M5Q (device 0) .....                                  | 52        |
| 3.6.3      | VNF9D1M5Q (device 1) .....                                  | 53        |
| 3.6.4      | VNF9D5SF .....                                              | 54        |
| 3.6.5      | VNF9Q20SF .....                                             | 55        |
| <b>3.7</b> | <b>P-channel companion chip functionality .....</b>         | <b>56</b> |
| 3.7.1      | Test conditions .....                                       | 56        |
| 3.7.2      | VNF9D1M5Q .....                                             | 57        |
| 3.7.3      | VNF9D5SF .....                                              | 60        |
| 3.7.4      | VNF9Q20SF .....                                             | 61        |
| <b>4</b>   | <b>Software description .....</b>                           | <b>62</b> |
| 4.1        | The AutoDevKit ecosystem .....                              | 62        |
| 4.2        | AEK-POW-PDUMINI software stack .....                        | 63        |
| 4.2.1      | AEK-POW-PDUMINI firmware basic startup steps .....          | 64        |
| 4.2.2      | Driver layer .....                                          | 64        |
| 4.2.3      | Service layer .....                                         | 66        |
| 4.2.4      | eFuse module implemented functions .....                    | 68        |
| 4.2.5      | eFuse module interaction with other modules .....           | 69        |
| 4.2.6      | e-fuse device configuration .....                           | 69        |
| 4.2.7      | e-fuse channel transition from Active to Standby mode ..... | 70        |
| 4.2.8      | PMIC_mng of the service layer .....                         | 71        |
| 4.2.9      | COMM_mng of the service layer .....                         | 72        |
| 4.2.10     | CAN protocol usage .....                                    | 73        |
| 4.2.11     | e-fuse channel activation example via CAN protocol .....    | 74        |
| 4.2.12     | ASS_mng of the service layer .....                          | 75        |
| 4.2.13     | PARK_mng of the service layer .....                         | 76        |
| 4.2.14     | ERR_mng of the service layer .....                          | 77        |
| 4.2.15     | List of diagnosable errors for ERR_mng .....                | 78        |
| 4.2.16     | UDS_mng of the service layer .....                          | 79        |

|                   |        |                                                                        |            |
|-------------------|--------|------------------------------------------------------------------------|------------|
|                   | 4.2.17 | UDS sessions, SIDs, and DIDs .....                                     | 80         |
| <b>5</b>          |        | <b>AEK-POW-PDUMINI software state machine .....</b>                    | <b>82</b>  |
|                   | 5.1    | Normal mode .....                                                      | 82         |
|                   | 5.2    | Parking mode .....                                                     | 83         |
|                   | 5.3    | Task scheduling and duration .....                                     | 84         |
| <b>6</b>          |        | <b>Demos available in AutoDevKit .....</b>                             | <b>89</b>  |
|                   | 6.1    | SPC58ECxx_RLA AEK_POW_PDUMINI demo .....                               | 89         |
|                   | 6.1.1  | UDS service manager .....                                              | 91         |
|                   | 6.1.2  | PMIC service manager .....                                             | 91         |
|                   | 6.1.3  | e-fuse service manager .....                                           | 91         |
|                   | 6.1.4  | Park service manager .....                                             | 92         |
|                   | 6.1.5  | ERR service manager .....                                              | 92         |
|                   | 6.1.6  | COMM service manager .....                                             | 92         |
|                   | 6.1.7  | ASS service manager .....                                              | 92         |
|                   | 6.1.8  | Operational flow .....                                                 | 92         |
|                   | 6.2    | Additional demos .....                                                 | 92         |
|                   | 6.2.1  | Multi-Unit Power Distribution via CAN Bus .....                        | 93         |
|                   | 6.2.2  | Remote Multi-Gateway Power Distribution Management over Ethernet ..... | 93         |
|                   | 6.3    | How to import a demo from AutoDevKit .....                             | 94         |
| <b>7</b>          |        | <b>Schematic diagrams .....</b>                                        | <b>96</b>  |
| <b>8</b>          |        | <b>Bill of materials .....</b>                                         | <b>106</b> |
| <b>9</b>          |        | <b>Board versions .....</b>                                            | <b>113</b> |
| <b>10</b>         |        | <b>Regulatory compliance information .....</b>                         | <b>114</b> |
| <b>Appendix A</b> |        | <b>UDS extensions and threshold configuration .....</b>                | <b>115</b> |
|                   | A.1    | Supported SID .....                                                    | 115        |
|                   | A.2    | Data identifiers added to the existing set .....                       | 115        |
|                   | A.3    | Request format .....                                                   | 115        |
|                   | A.4    | Channel mapping (AEK-POW-PDUMINI ↔ devices) .....                      | 116        |
|                   | A.4.1  | Channel mapping .....                                                  | 116        |
|                   | A.4.2  | EEPROM layout and handling .....                                       | 117        |
|                   | A.5    | Device configuration functions .....                                   | 117        |
|                   | A.6    | L99SP08 threshold settings .....                                       | 117        |
|                   | A.7    | VNF9D1M5Q I <sup>2</sup> t settings .....                              | 117        |
|                   | A.8    | VNF9D5SF I <sup>2</sup> t settings .....                               | 117        |
|                   | A.9    | VNF9Q20SF I <sup>2</sup> t settings .....                              | 118        |
| <b>Appendix B</b> |        | <b>UDS usage examples .....</b>                                        | <b>119</b> |

|                               |                                                               |            |
|-------------------------------|---------------------------------------------------------------|------------|
| <b>B.1</b>                    | Switch from Standard Session to Extended Session .....        | 119        |
| <b>B.2</b>                    | Set L99SP08 I <sub>THR</sub> threshold using UDS.....         | 119        |
| <b>B.3</b>                    | Channel mapping for L99SP08 .....                             | 120        |
| <b>B.4</b>                    | Setting I <sup>2</sup> t VNF9D1M5Q thresholds using UDS.....  | 121        |
| <b>B.5</b>                    | Setting VNF9D5SF I <sup>2</sup> t thresholds using UDS .....  | 124        |
| <b>B.6</b>                    | Setting VNF9Q20SF I <sup>2</sup> t thresholds using UDS ..... | 126        |
| <b>Revision history .....</b> |                                                               | <b>130</b> |
| <b>List of tables .....</b>   |                                                               | <b>135</b> |
| <b>List of figures.....</b>   |                                                               | <b>136</b> |



## List of tables

|                  |                                                                        |     |
|------------------|------------------------------------------------------------------------|-----|
| <b>Table 1.</b>  | Output connectors and related output values . . . . .                  | 10  |
| <b>Table 2.</b>  | Used tools . . . . .                                                   | 22  |
| <b>Table 3.</b>  | Results' summary . . . . .                                             | 43  |
| <b>Table 4.</b>  | Device-dependent channel parameters . . . . .                          | 69  |
| <b>Table 5.</b>  | Supported CAN messages and IDs . . . . .                               | 74  |
| <b>Table 6.</b>  | ASS module functions . . . . .                                         | 76  |
| <b>Table 7.</b>  | Summary table for ERR interactions . . . . .                           | 78  |
| <b>Table 8.</b>  | ERR diagnosable errors . . . . .                                       | 78  |
| <b>Table 9.</b>  | SID codes . . . . .                                                    | 80  |
| <b>Table 10.</b> | Examples of DID codes . . . . .                                        | 80  |
| <b>Table 11.</b> | UDS message examples . . . . .                                         | 81  |
| <b>Table 12.</b> | AEK-POW-PDUMINI bill of materials . . . . .                            | 106 |
| <b>Table 13.</b> | AEK-POW-PDUMINI versions . . . . .                                     | 113 |
| <b>Table 14.</b> | Mapping between onboard devices and AEK-POW-PDUMINI channels . . . . . | 116 |
| <b>Table 15.</b> | Extended diagnostic session details . . . . .                          | 119 |
| <b>Table 16.</b> | Current threshold settings for L99SP08 . . . . .                       | 119 |
| <b>Table 17.</b> | Current threshold settings for L99SP08: field description . . . . .    | 120 |
| <b>Table 18.</b> | L99SP08 channel mapping . . . . .                                      | 120 |
| <b>Table 19.</b> | Threshold bytes and related L99SP08 current threshold . . . . .        | 120 |
| <b>Table 20.</b> | Current threshold settings for VNF9D1M5Q . . . . .                     | 121 |
| <b>Table 21.</b> | Current threshold settings for VNF9D1M5Q : field description . . . . . | 121 |
| <b>Table 22.</b> | AEK-POW-PDUMINI and VNF9D1M5Q channel mapping . . . . .                | 121 |
| <b>Table 23.</b> | I <sub>NOM</sub> value table . . . . .                                 | 122 |
| <b>Table 24.</b> | T <sub>NOM</sub> value table . . . . .                                 | 123 |
| <b>Table 25.</b> | I <sub>PEAK</sub> value table . . . . .                                | 123 |
| <b>Table 26.</b> | Example request frame . . . . .                                        | 124 |
| <b>Table 27.</b> | Field meaning . . . . .                                                | 124 |
| <b>Table 28.</b> | AEK-POW-PDUMINI and VNF9D5SF channel mapping . . . . .                 | 124 |
| <b>Table 29.</b> | I <sub>NOM</sub> value table . . . . .                                 | 125 |
| <b>Table 30.</b> | T <sub>NOM</sub> value table . . . . .                                 | 125 |
| <b>Table 31.</b> | I <sub>SHUTOFF</sub> value table . . . . .                             | 126 |
| <b>Table 32.</b> | Example request frame . . . . .                                        | 127 |
| <b>Table 33.</b> | Field meaning . . . . .                                                | 127 |
| <b>Table 34.</b> | AEK-POW-PDUMINI and VNF9Q20SF channel mapping . . . . .                | 127 |
| <b>Table 35.</b> | I <sub>NOM</sub> value table . . . . .                                 | 128 |
| <b>Table 36.</b> | <b>T<sub>NOM</sub> value table</b> . . . . .                           | 128 |
| <b>Table 37.</b> | <b>I<sub>SHUTOFF</sub> value table</b> . . . . .                       | 129 |
| <b>Table 38.</b> | Document revision history . . . . .                                    | 130 |

## List of figures

|                   |                                                                                                                                            |    |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------|----|
| <b>Figure 1.</b>  | Example of a possible vehicle power distribution system . . . . .                                                                          | 1  |
| <b>Figure 2.</b>  | AEK-POW-PDUMINI top side view . . . . .                                                                                                    | 2  |
| <b>Figure 3.</b>  | AEK-POW-PDUMINI conceptual block diagram . . . . .                                                                                         | 3  |
| <b>Figure 4.</b>  | AEK-POW-PDUMINI functional blocks. . . . .                                                                                                 | 5  |
| <b>Figure 5.</b>  | AEK-POW-PDUMINI - schematic view. . . . .                                                                                                  | 6  |
| <b>Figure 6.</b>  | 96-pin automotive unified connector - board view . . . . .                                                                                 | 8  |
| <b>Figure 7.</b>  | 96-pin automotive unified connector - schematic view . . . . .                                                                             | 8  |
| <b>Figure 8.</b>  | AEK-POW-PDUMINI main MCU connector pinout. . . . .                                                                                         | 9  |
| <b>Figure 9.</b>  | AEK-MCU-C4MLIT3 main MCU board connected to AEK-POW-PDUMINI PDU board. . . . .                                                             | 9  |
| <b>Figure 10.</b> | Power input and output connectors, board view . . . . .                                                                                    | 10 |
| <b>Figure 11.</b> | CAN connector - board view. . . . .                                                                                                        | 11 |
| <b>Figure 12.</b> | AEK-POW-PDUMINI Assembly mode emulation - schematic and board view . . . . .                                                               | 11 |
| <b>Figure 13.</b> | AEK-POW-PDUMINI status LEDs - schematic view. . . . .                                                                                      | 12 |
| <b>Figure 14.</b> | L99SP08 and LDL40 board view and status LEDs: ENx, LH, Dlx, and DIAG. . . . .                                                              | 13 |
| <b>Figure 15.</b> | L99SP08 and LDL40 - schematic view. . . . .                                                                                                | 14 |
| <b>Figure 16.</b> | VNF9D1M5Q - schematic view . . . . .                                                                                                       | 16 |
| <b>Figure 17.</b> | VNF9D1M5Q (device 0 e 1) board view and status LEDs: OUTx, STDBY, LH, EMx, and DIAGx . . . . .                                             | 16 |
| <b>Figure 18.</b> | VNF9D5SF - schematic view . . . . .                                                                                                        | 17 |
| <b>Figure 19.</b> | VNF9D5SF board view and output status LEDs: OUT4 and OUT5 . . . . .                                                                        | 17 |
| <b>Figure 20.</b> | VNF9Q20SF - schematic view . . . . .                                                                                                       | 18 |
| <b>Figure 21.</b> | VNF9Q20SF board view and output status LEDs: OUT6, OUT7, OUT8, OUT9 . . . . .                                                              | 18 |
| <b>Figure 22.</b> | SPSB081 - board view. . . . .                                                                                                              | 19 |
| <b>Figure 23.</b> | SPSB081 - schematic view . . . . .                                                                                                         | 20 |
| <b>Figure 24.</b> | VN7140AJ - board view . . . . .                                                                                                            | 20 |
| <b>Figure 25.</b> | VN7140AJ - schematic view . . . . .                                                                                                        | 21 |
| <b>Figure 26.</b> | OUT0 wake-up, DIAG_SP08 signal low and CSN0 and CSN1 reach 5 V ( $t_{CSN0-1}(5\text{ V}) = 211.3\text{ ms}$ ) . . . . .                    | 23 |
| <b>Figure 27.</b> | OUT0 wake-up, CSN0 and CSN1 reach 5 V and CSN0 start communication . . . . .                                                               | 23 |
| <b>Figure 28.</b> | OUT0 wake-up signals detail ( $t_{DIAG\_SP08(low)} = 300\text{ }\mu\text{s}$ $t_{V1(high)} = 500\text{ }\mu\text{s}$ ) . . . . .           | 24 |
| <b>Figure 29.</b> | OUT2 wake-up - DIAG_SP08, CSN0, CSN1, and output signals . . . . .                                                                         | 24 |
| <b>Figure 30.</b> | OUT4 wake-up ( $t_{CSN0-1}(5\text{ V}) = 210.43\text{ ms}$ ) . . . . .                                                                     | 25 |
| <b>Figure 31.</b> | OUT5 wake-up . . . . .                                                                                                                     | 25 |
| <b>Figure 32.</b> | Module wake-up through CAN ( $t_{CSN0-1}(5\text{ V}) = 210.73\text{ ms}$ ) . . . . .                                                       | 26 |
| <b>Figure 33.</b> | CAN wake-up signals detail . . . . .                                                                                                       | 26 |
| <b>Figure 34.</b> | CAN wake-up, CSN0 and CSN1 reach 5 V and CSN0 startup communication . . . . .                                                              | 27 |
| <b>Figure 35.</b> | OUT0, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ , $t_{nom} = 1\text{ s}$ ) (1 of 3) . . . . .   | 28 |
| <b>Figure 36.</b> | OUT0, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ , $t_{nom} = 1\text{ s}$ ) (2 of 3) . . . . .   | 28 |
| <b>Figure 37.</b> | OUT0, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ , $t_{nom} = 1\text{ s}$ ) (3 of 3) . . . . .   | 29 |
| <b>Figure 38.</b> | OUT1, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ , $t_{nom} = 172\text{ s}$ ) (1 of 3) . . . . . | 29 |
| <b>Figure 39.</b> | OUT1, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ , $t_{nom} = 172\text{ s}$ ) (2 of 3) . . . . . | 30 |
| <b>Figure 40.</b> | OUT1, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ , $t_{nom} = 172\text{ s}$ ) (3 of 3) . . . . . | 30 |
| <b>Figure 41.</b> | OUT2, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ , $t_{nom} = 1\text{ s}$ ) (1 of 3) . . . . .   | 31 |
| <b>Figure 42.</b> | OUT2, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ , $t_{nom} = 1\text{ s}$ ) (2 of 3) . . . . .   | 31 |
| <b>Figure 43.</b> | OUT2, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 21\text{ A}$ , $t_{nom} = 1\text{ s}$ ) (3 of 3) . . . . .   | 32 |
| <b>Figure 44.</b> | OUT3, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ , $t_{nom} = 172\text{ s}$ ) (1 of 3) . . . . . | 32 |
| <b>Figure 45.</b> | OUT3, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ , $t_{nom} = 172\text{ s}$ ) (2 of 3) . . . . . | 33 |
| <b>Figure 46.</b> | OUT3, VNF9D1M5Q STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 16\text{ A}$ , $t_{nom} = 172\text{ s}$ ) (3 of 3) . . . . . | 33 |
| <b>Figure 47.</b> | OUT4, VNF9D5SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 4\text{ A}$ , $t_{nom} = 44\text{ s}$ ) (1 of 3) . . . . .    | 34 |
| <b>Figure 48.</b> | OUT4, VNF9D5SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 4\text{ A}$ , $t_{nom} = 44\text{ s}$ ) (2 of 3) . . . . .    | 34 |
| <b>Figure 49.</b> | OUT4, VNF9D5SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 4\text{ A}$ , $t_{nom} = 44\text{ s}$ ) (3 of 3) . . . . .    | 35 |

|                    |                                                                                                                                                       |    |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| <b>Figure 50.</b>  | OUT5, VNF9D5SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 10\text{ A}$ , $t_{nom} = 214\text{ s}$ ) (1 of 3).                      | 35 |
| <b>Figure 51.</b>  | OUT5, VNF9D5SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 10\text{ A}$ , $t_{nom} = 214\text{ s}$ ) (2 of 3).                      | 36 |
| <b>Figure 52.</b>  | OUT5, VNF9D5SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 10\text{ A}$ , $t_{nom} = 214\text{ s}$ ) (3 of 3).                      | 36 |
| <b>Figure 53.</b>  | OUT6, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (1 of 3).                      | 37 |
| <b>Figure 54.</b>  | OUT6, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (2 of 3).                      | 37 |
| <b>Figure 55.</b>  | OUT6, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (3 of 3).                      | 38 |
| <b>Figure 56.</b>  | OUT7, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (1 of 3).                      | 38 |
| <b>Figure 57.</b>  | OUT7, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (2 of 3).                      | 39 |
| <b>Figure 58.</b>  | OUT7, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (3 of 3).                      | 39 |
| <b>Figure 59.</b>  | OUT8, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (1 of 3).                      | 40 |
| <b>Figure 60.</b>  | OUT8, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (2 of 3).                      | 40 |
| <b>Figure 61.</b>  | OUT8, VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (3 of 3).                      | 41 |
| <b>Figure 62.</b>  | OUT9 (@15 A load current), VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (1 of 3). | 41 |
| <b>Figure 63.</b>  | OUT9 (@15 A load current), VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (2 of 3). | 42 |
| <b>Figure 64.</b>  | OUT9 (@15 A load current), VNF9Q20SF STi <sup>2</sup> Fuse regular functionality test ( $I_{nom} = 6\text{ A}$ , $t_{nom} = 300\text{ s}$ ) (3 of 3). | 42 |
| <b>Figure 65.</b>  | VNF9D1M5Q @ 26 A (OUT0) - Thermal image: 5 secs                                                                                                       | 43 |
| <b>Figure 66.</b>  | VNF9D1M5Q @ 26 A (OUT0) - Thermal image: 160 secs                                                                                                     | 43 |
| <b>Figure 67.</b>  | VNF9D1M5Q @ 26 A (OUT0) - Thermal image: 300 secs                                                                                                     | 44 |
| <b>Figure 68.</b>  | VNF9D1M5Q @ 26 A (OUT1) - Thermal image: 5 secs                                                                                                       | 44 |
| <b>Figure 69.</b>  | VNF9D1M5Q @ 26 A (OUT1) - Thermal image: 160 secs                                                                                                     | 44 |
| <b>Figure 70.</b>  | VNF9D1M5Q @ 26 A (OUT1) - Thermal image: 300 secs                                                                                                     | 44 |
| <b>Figure 71.</b>  | VNF9D1M5Q @ 25 A (OUT2) - Thermal image: 5 secs                                                                                                       | 45 |
| <b>Figure 72.</b>  | VNF9D1M5Q @ 25 A (OUT2) - Thermal image: 160 secs                                                                                                     | 45 |
| <b>Figure 73.</b>  | VNF9D1M5Q @ 25 A (OUT2) - Thermal image: 300 secs                                                                                                     | 45 |
| <b>Figure 74.</b>  | VNF9D1M5Q @ 25 A (OUT3) - Thermal image: 5 secs                                                                                                       | 45 |
| <b>Figure 75.</b>  | VNF9D1M5Q @ 25 A (OUT3) - Thermal image: 160 secs                                                                                                     | 45 |
| <b>Figure 76.</b>  | VNF9D1M5Q @ 25 A (OUT3) - Thermal image: 300 secs                                                                                                     | 46 |
| <b>Figure 77.</b>  | VNF9D5SF @ 14 A (OUT4) - Thermal image: 5 secs                                                                                                        | 46 |
| <b>Figure 78.</b>  | VNF9D5SF @ 14 A (OUT4) - Thermal image: 160 secs                                                                                                      | 46 |
| <b>Figure 79.</b>  | VNF9D5SF @ 14 A (OUT4) - Thermal image: 300 secs                                                                                                      | 46 |
| <b>Figure 80.</b>  | VNF9D5SF @ 14 A (OUT5) - Thermal image: 5 secs                                                                                                        | 47 |
| <b>Figure 81.</b>  | VNF9D5SF @ 14 A (OUT5) - Thermal image: 160 secs                                                                                                      | 47 |
| <b>Figure 82.</b>  | VNF9D5SF @ 14 A (OUT5) - Thermal image: 300 secs                                                                                                      | 47 |
| <b>Figure 83.</b>  | VNF9Q20SF @ 5 A (OUT6) - Thermal image: 5 secs                                                                                                        | 47 |
| <b>Figure 84.</b>  | VNF9Q20SF @ 5 A (OUT6) - Thermal image: 160 secs                                                                                                      | 47 |
| <b>Figure 85.</b>  | VNF9Q20SF @ 5 A (OUT6) - Thermal image: 300 secs                                                                                                      | 48 |
| <b>Figure 86.</b>  | VNF9Q20SF @ 5 A (OUT7) - Thermal image: 5 secs                                                                                                        | 48 |
| <b>Figure 87.</b>  | VNF9Q20SF @ 5 A (OUT7) - Thermal image: 160 secs                                                                                                      | 48 |
| <b>Figure 88.</b>  | VNF9Q20SF @ 5 A (OUT7) - Thermal image: 300 secs                                                                                                      | 48 |
| <b>Figure 89.</b>  | VNF9Q20SF @ 5 A (OUT8) - Thermal image: 5 secs                                                                                                        | 49 |
| <b>Figure 90.</b>  | VNF9Q20SF @ 5 A (OUT8) - Thermal image: 160 secs                                                                                                      | 49 |
| <b>Figure 91.</b>  | VNF9Q20SF @ 5 A (OUT8) - Thermal image: 300 secs                                                                                                      | 49 |
| <b>Figure 92.</b>  | VNF9Q20SF @ 5 A (OUT9) - Thermal image: 5 secs                                                                                                        | 49 |
| <b>Figure 93.</b>  | VNF9Q20SF @ 5 A (OUT9) - Thermal image: 160 secs                                                                                                      | 49 |
| <b>Figure 94.</b>  | VNF9Q20SF @ 5 A (OUT9) - Thermal image: 300 secs                                                                                                      | 50 |
| <b>Figure 95.</b>  | VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1) - Thermal image: 5 secs                                                                                             | 50 |
| <b>Figure 96.</b>  | VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1) - Thermal image: 160 secs                                                                                           | 50 |
| <b>Figure 97.</b>  | VNF9D1M5Q @ 24 A+24 A (OUT0+OUT1) - Thermal image: 300 secs                                                                                           | 51 |
| <b>Figure 98.</b>  | VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3) - Thermal image: 5 secs                                                                                             | 51 |
| <b>Figure 99.</b>  | VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3) - Thermal image: 160 secs                                                                                           | 51 |
| <b>Figure 100.</b> | VNF9D1M5Q @ 24 A+24 A (OUT2+OUT3) - Thermal image: 300 secs                                                                                           | 51 |
| <b>Figure 101.</b> | VNF9D5SF @ 14 A+14 A (OUT4+OUT5) - Thermal image: 5 secs                                                                                              | 52 |

|                    |                                                                                             |    |
|--------------------|---------------------------------------------------------------------------------------------|----|
| <b>Figure 102.</b> | VNF9D5SF @ 14 A+14 A (OUT4+OUT5) - Thermal image: 160 secs                                  | 52 |
| <b>Figure 103.</b> | VNF9D5SF @ 14 A+14 A (OUT4+OUT5) - Thermal image: 300 secs                                  | 52 |
| <b>Figure 104.</b> | VNF9D1M5Q (dev0), OUT0 and OUT1 independent channel operation                               | 53 |
| <b>Figure 105.</b> | VNF9D1M5Q (dev0), OUT0 and OUT1 independent channel operation (details of latch off time)   | 53 |
| <b>Figure 106.</b> | VNF9D1M5Q (dev1), OUT2 and OUT3 independent channel operation                               | 54 |
| <b>Figure 107.</b> | VNF9D1M5Q (dev1), OUT2 and OUT3 independent channel operation (details of latch off time)   | 54 |
| <b>Figure 108.</b> | VNF9D5SF, OUT4 and OUT5 independent channel operation                                       | 55 |
| <b>Figure 109.</b> | VNF9D5SF, OUT4 and OUT5 independent channel operation (details of latch off time)           | 55 |
| <b>Figure 110.</b> | VNF9Q20SF, OUT6 and OUT7 independent channel operation                                      | 56 |
| <b>Figure 111.</b> | VNF9Q20SF, OUT6 and OUT7 independent channel operation (details of latch off time)          | 56 |
| <b>Figure 112.</b> | OUT0, P-channel companion chip activating VNF9D1M5Q device in Normal mode                   | 57 |
| <b>Figure 113.</b> | OUT0, P-channel companion chip activating VNF9D1M5Q device in Parking mode                  | 57 |
| <b>Figure 114.</b> | OUT1, P-channel companion chip activating VNF9D1M5Q device                                  | 58 |
| <b>Figure 115.</b> | OUT2, P-channel companion chip activating VNF9D1M5Q device                                  | 58 |
| <b>Figure 116.</b> | OUT3, P-channel companion chip activating VNF9D1M5Q device                                  | 59 |
| <b>Figure 117.</b> | OUT4, P-channel companion chip activating VNF9D5SF device                                   | 60 |
| <b>Figure 118.</b> | OUT5, P-channel companion chip activating VNF9D5SF device                                   | 60 |
| <b>Figure 119.</b> | OUT6, P-channel companion chip activating VNF9Q20SF device                                  | 61 |
| <b>Figure 120.</b> | OUT7, P-channel companion chip activating VNF9Q20SF device                                  | 61 |
| <b>Figure 121.</b> | Example of fast prototyping using AutoDevKit                                                | 62 |
| <b>Figure 122.</b> | SPC8 HAL portion                                                                            | 63 |
| <b>Figure 123.</b> | Portion of the device layer dedicated to e-fuse devices                                     | 63 |
| <b>Figure 124.</b> | Three-layer stack for L99SP08 in driver layer                                               | 65 |
| <b>Figure 125.</b> | Nested function calls from service layer to MCU peripheral for L99SP08 Channel 0 activation | 65 |
| <b>Figure 126.</b> | Service layer key components                                                                | 66 |
| <b>Figure 127.</b> | Example of channel enablement request                                                       | 68 |
| <b>Figure 128.</b> | eFuse module functions                                                                      | 68 |
| <b>Figure 129.</b> | eFuse module diagnostics and monitoring functions                                           | 69 |
| <b>Figure 130.</b> | Example of communication with PMIC module                                                   | 71 |
| <b>Figure 131.</b> | PMIC module functions                                                                       | 72 |
| <b>Figure 132.</b> | COMM module functions                                                                       | 73 |
| <b>Figure 133.</b> | Example code of how to turn on an e-fuse channel via CAN                                    | 74 |
| <b>Figure 134.</b> | Example code of message dispatched to the eFuse module                                      | 75 |
| <b>Figure 135.</b> | Park module functions                                                                       | 77 |
| <b>Figure 136.</b> | ERR module functions                                                                        | 78 |
| <b>Figure 137.</b> | UDS module functions                                                                        | 79 |
| <b>Figure 138.</b> | Application software state machine                                                          | 82 |
| <b>Figure 139.</b> | eFuse task timing diagram                                                                   | 84 |
| <b>Figure 140.</b> | PMIC task timing diagram                                                                    | 85 |
| <b>Figure 141.</b> | ERR task timing diagram                                                                     | 85 |
| <b>Figure 142.</b> | COMMRx task timing diagram                                                                  | 86 |
| <b>Figure 143.</b> | Park task timing diagram                                                                    | 87 |
| <b>Figure 144.</b> | Sti <sup>2</sup> Fuse PDUB software timing                                                  | 87 |
| <b>Figure 145.</b> | All task timing diagram                                                                     | 88 |
| <b>Figure 146.</b> | SPC58ECxx_RLA AEK_POW_PDUMINI in AutoDevKit                                                 | 89 |
| <b>Figure 147.</b> | Demo components folder                                                                      | 90 |
| <b>Figure 148.</b> | Source folder                                                                               | 90 |
| <b>Figure 149.</b> | main.c file                                                                                 | 91 |
| <b>Figure 150.</b> | Importing the demo from the application library (1 of 4)                                    | 94 |
| <b>Figure 151.</b> | Importing the demo from the application library (2 of 4)                                    | 94 |
| <b>Figure 152.</b> | Importing the demo from the application library (3 of 4)                                    | 95 |
| <b>Figure 153.</b> | Importing the demo from the application library (4 of 4)                                    | 95 |
| <b>Figure 154.</b> | AEK-POW-PDUMINI circuit schematic (1 of 10)                                                 | 96 |
| <b>Figure 155.</b> | AEK-POW-PDUMINI circuit schematic (2 of 10)                                                 | 97 |
| <b>Figure 156.</b> | AEK-POW-PDUMINI circuit schematic (3 of 10)                                                 | 98 |

|                    |                                                                    |     |
|--------------------|--------------------------------------------------------------------|-----|
| <b>Figure 157.</b> | AEK-POW-PDUMINI circuit schematic (4 of 10) . . . . .              | 99  |
| <b>Figure 158.</b> | AEK-POW-PDUMINI circuit schematic (5 of 10) . . . . .              | 100 |
| <b>Figure 159.</b> | AEK-POW-PDUMINI circuit schematic (6 of 10) . . . . .              | 101 |
| <b>Figure 160.</b> | AEK-POW-PDUMINI circuit schematic (7 of 10) . . . . .              | 102 |
| <b>Figure 161.</b> | AEK-POW-PDUMINI circuit schematic (8 of 10) . . . . .              | 103 |
| <b>Figure 162.</b> | AEK-POW-PDUMINI circuit schematic (9 of 10) . . . . .              | 104 |
| <b>Figure 163.</b> | AEK-POW-PDUMINI circuit schematic (10 of 10) . . . . .             | 105 |
| <b>Figure 164.</b> | Extended diagnostic session: data byte view . . . . .              | 119 |
| <b>Figure 165.</b> | Current threshold settings for L99SP08: data byte view . . . . .   | 120 |
| <b>Figure 166.</b> | Current threshold settings for VNF9D1M5Q: data byte view . . . . . | 121 |
| <b>Figure 167.</b> | Example request frame: data byte view . . . . .                    | 124 |
| <b>Figure 168.</b> | Example request frame: data byte view . . . . .                    | 127 |

**IMPORTANT NOTICE – READ CAREFULLY**

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice.

In the event of any conflict between the provisions of this document and the provisions of any contractual arrangement in force between the purchasers and ST, the provisions of such contractual arrangement shall prevail.

The purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgment.

The purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of the purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

If the purchasers identify an ST product that meets their functional and performance requirements but that is not designated for the purchasers' market segment, the purchasers shall contact ST for more information.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to [www.st.com/trademarks](http://www.st.com/trademarks). All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2026 STMicroelectronics – All rights reserved