UM12013

EVSE-SIG-BRD1X User Manual

Rev. 1 — 18 June 2024 User manual

Document information

Information	Content
Keywords	UM12013, EVSE-SIG-BRD1X, LPC5536, LPC55S36, SJA1110B, CG5317, HPGP, EVSE, EV, control pilot, proximity pilot
Abstract	The EVSE-SIG-BRD1X is an add-on development board that supports electric vehicle supply equipment (EVSE) or electric vehicle (EV) platform development.

1 Board overview

EVSE-SIG-BRD1X is an add-on development board that supports electric vehicle supply equipment (EVSE) or electric vehicle (EV) platform development. The main host of the system is on a separate processor development board, for example, NXP i.MX RT1060 EVK, i.MX 8M Nano EVK, or S32G-VNP-RDB3. The ISO 15118 protocol stack and communication software run on the host processor. The power-line communication (PLC) path is via the onboard HomePlug Green PHY (HPGP) transceiver (Lumissil IS32CG5317). The EVSE development platform, including the host controller, EVSE-SIG-BRD1X, security and NFC modules, and NXP Kinetis KM3x family of metering microcontroller solutions can form the basis of a full electric vehicle charging station for quick system design and prototyping.

This document describes the features and hardware and software details of EVSE-SIG-BRD1X. It also explains how to use and interface the board with the host controller boards. The software implementation is based on the NXP MCUXpresso SDK. The hardware design files and software of the board can be downloaded and referenced.

Note: *Read EVSE-SIG-BRD1X User Guide (UG10109) before proceeding to read this document further. UG10109 provides details about usable EVSE-SIG-BRD1X hardware interfaces and tools used for software development.*

1.1 Block diagrams

EVSE-SIG-BRD1X is primarily designed to host:

- A HomePlug Green PHY (HPGP) for the ISO 15118-2/20 communication line.
- J1772 PWM signaling for the control pilot feature.

The board also supports the proximity pilot, ground fault circuit interrupter (GFCI), and relay drive features. To support these hardware features, the board circuit is designed with power supplies, MCUs, ASICs, HPGP, QSPI flash, and Ethernet switches interconnected to related hardware components of the board. EVSE-SIG-BRD1X provides multiple host connector options for connecting the main host controller board.

Figure 1 shows the EVSE-SIG-BRD1X system hardware block diagram.

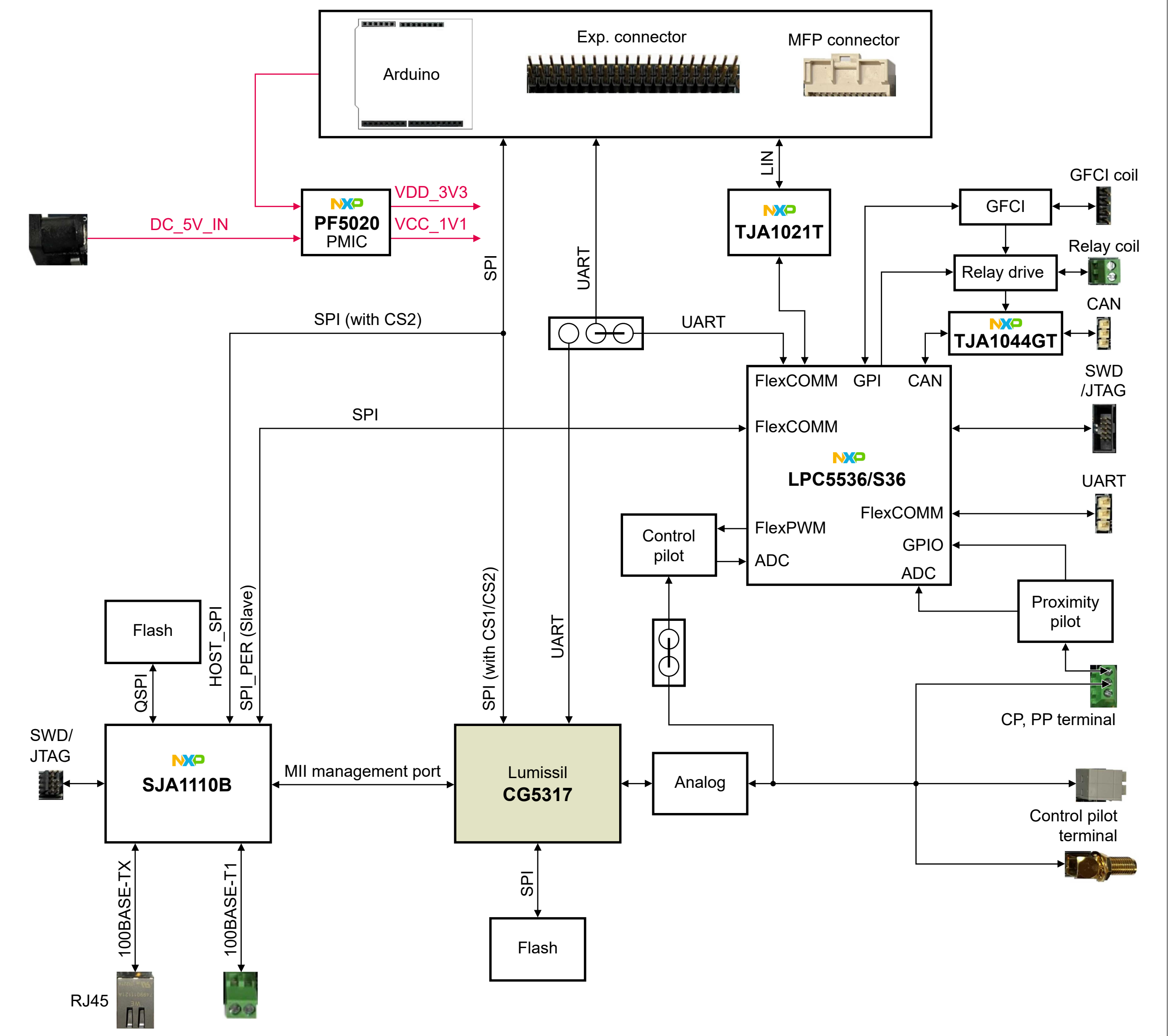

Figure 1. EVSE-SIG-BRD1X system hardware block diagram

Figure 2 shows the EVSE-SIG-BRD1X system software block diagram.

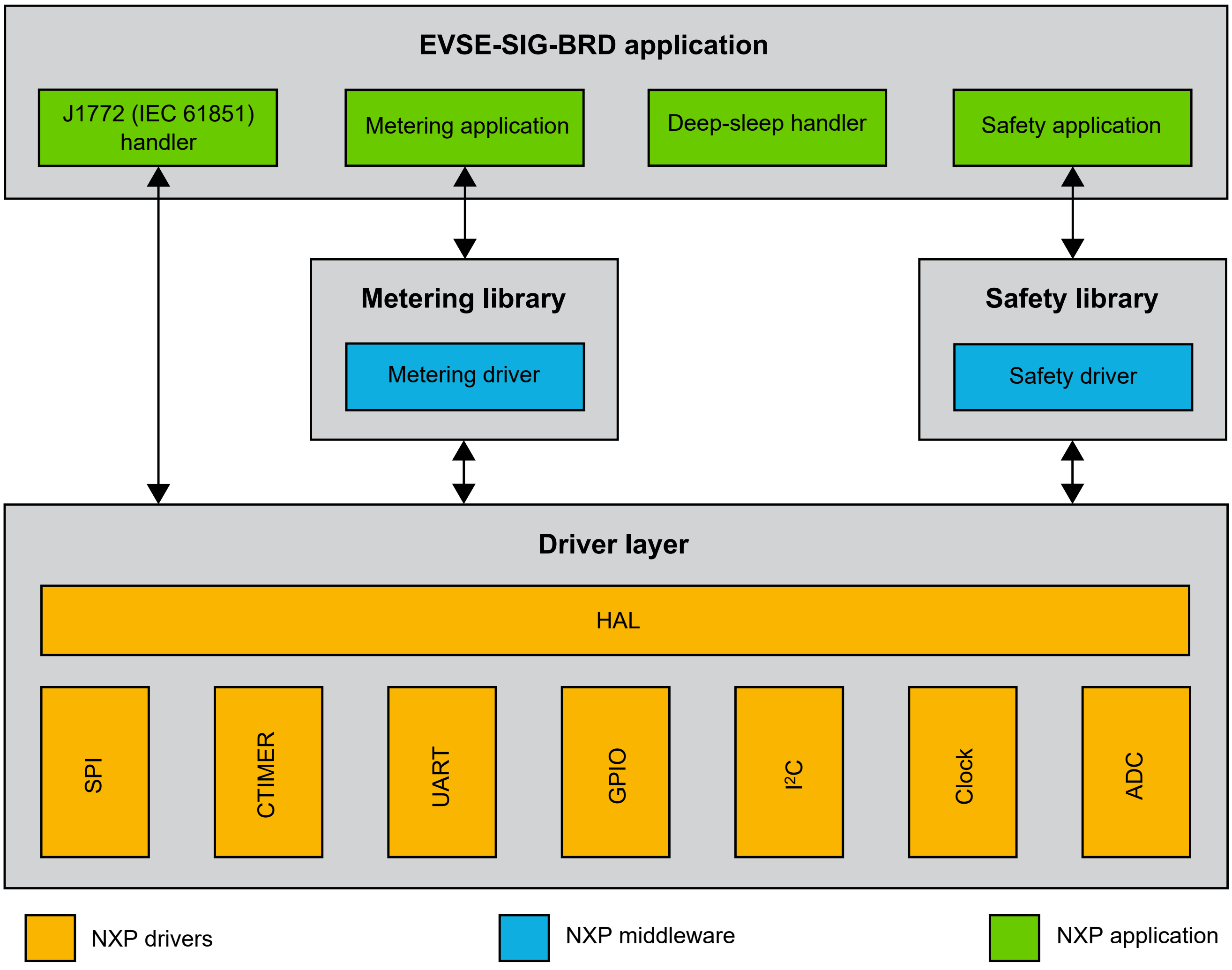

Figure 2. EVSE-SIG-BRD1X system software block diagram

1.2 Board features

Table 1 lists the board features of the EVSE-SIG-BRD1X.

Table 1. EVSE-SIG-BRD1X features

Board feature	Description
Embedded microcontroller	NXP LPC5536/LPC55S36 MCU, which features a 32-bit Arm Cortex-M33 core, 128 KB SRAM, 256 KB flash, FlexSPI with cache, USB FS, Flexcomm interface, CAN FD, 32-bit counters/timers, SCTimer/PWM, 16-bit 2.0 Msamples/s ADC, comparator, 12-bit DAC, op-amp, FlexPWM timer, QEI, temperature sensor, and CRC
Embedded HPGP	Lumissil CG5317
Embedded Ethernet switch MCU	NXP SJA1110B
Host connectors • Arduino socket connectors • EXP CN / GPIO header • MFP connector	• Power: +5 V, +3.3 V • One SPI port with two chip selects • One UART port • GPIOs • LIN (MFP only)

Table 1. EVSE-SIG-BRD1X features*...continued*

Board feature	Description
Ethernet host interface	• One 100BASE-T1 port • One 100BASE-TX port
CAN interface	One NXP TJA1044GT CAN PHY
LIN interface	One NXP TJA1021T/20/C LIN PHY
Debug interface	• Auxiliary UART port from LPC5536/LPC55S36 • SWD debug port of LPC5536/LPC55S36 for development
Control pilot	J1772 (IEC 61851) PWM, ISO 15118-2/20 EVSE and EV support
Proximity pilot	J1772 support
GFCI	GFCI detection and relay asynchronous triggering
Relay driver	Drive up to two DC coil relays at 12 V, 140 mA
Power	• Primary power supply options: – 5 V external power through DC power jack (J1) – Power from the host controller board through a host connector (Arduino / EXP CN / MFP) • Onboard +5 V to +12 V boost converter • Onboard +12 V to -12 V charge pump inverter
PCB	6.4 inch x 3 inch, 6-layer
Orderable part number	EVSE-SIG-BRD1X

1.3 Board pictures

Figure 3 shows the top-side view of EVSE-SIG-BRD1X.

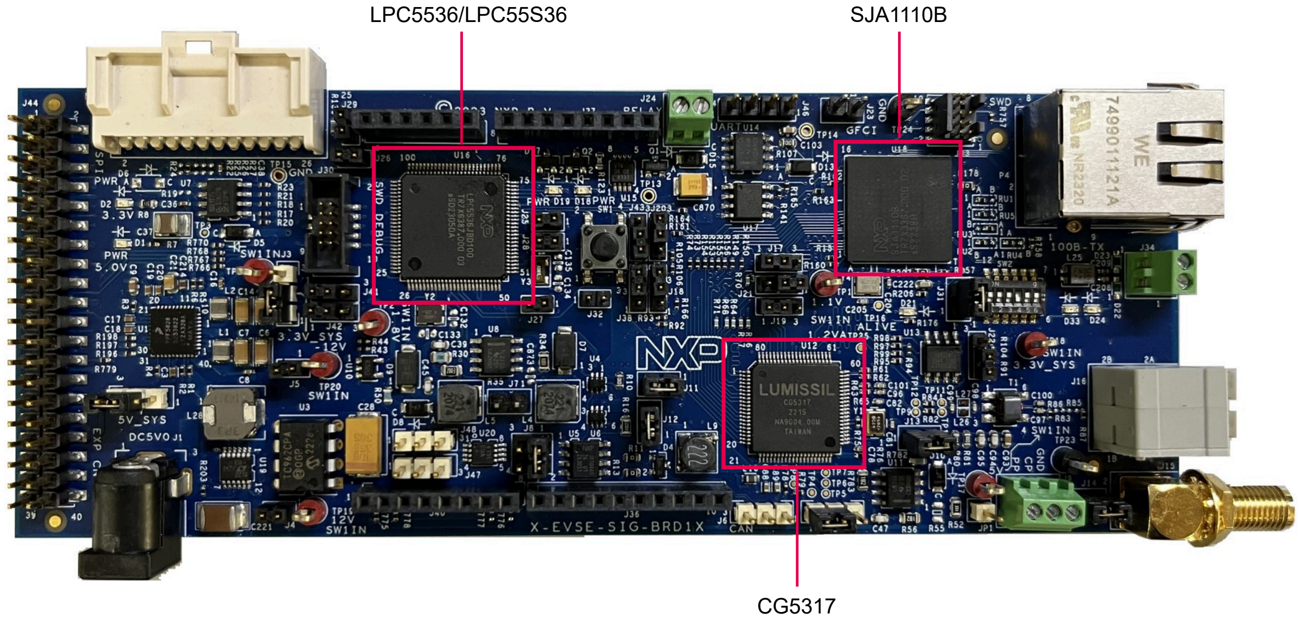

Figure 3. EVSE-SIG-BRD1X top-side view

Figure 4 shows the bottom-side view of EVSE-SIG-BRD1X.

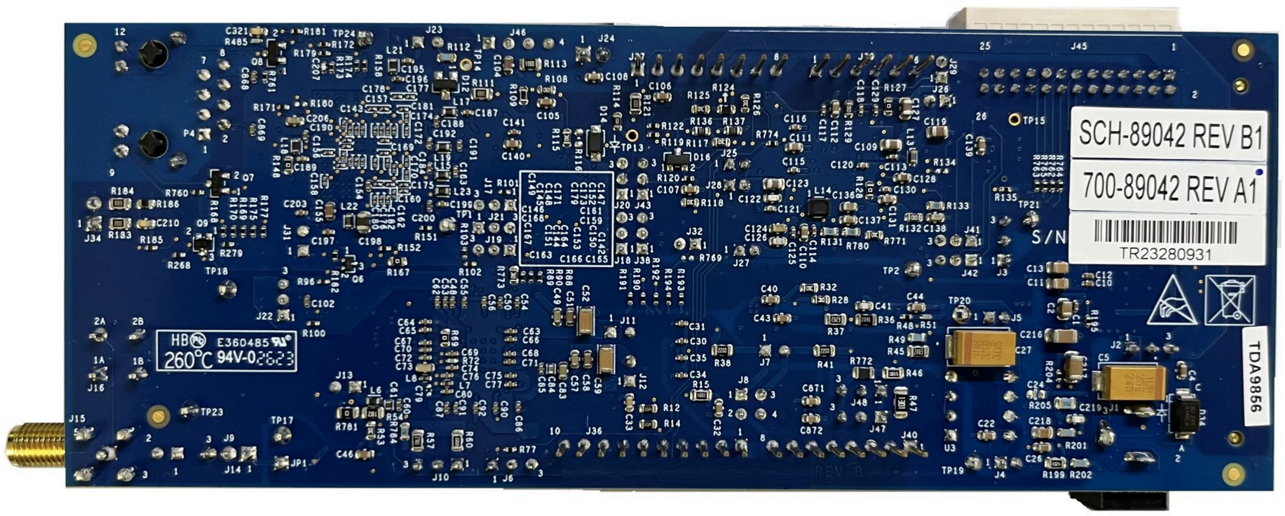

Figure 4. EVSE-SIG-BRD1X bottom-side view

1.4 Connectors

Figure 5 shows the EVSE-SIG-BRD1X connectors.

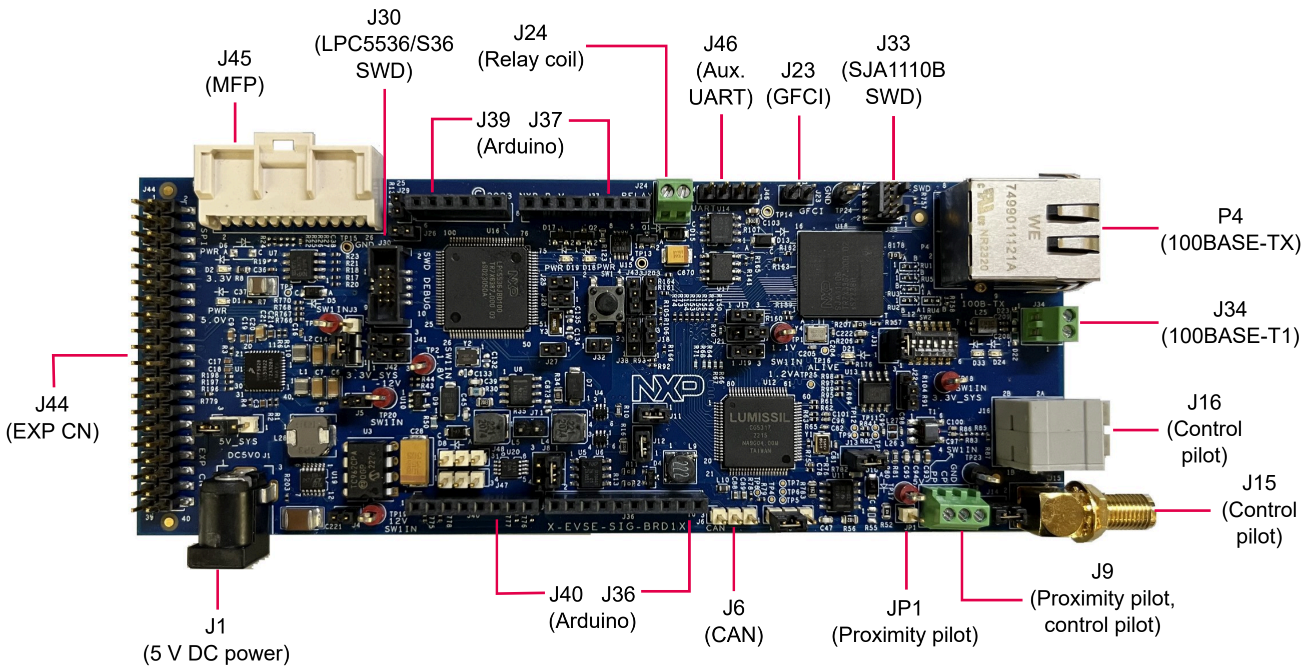

Figure 5. Connectors

Table 2 describes the connectors available on EVSE-SIG-BRD1X.

Table 2. EVSE-SIG-BRD1X connectors

Part identifier	PCB label	Connector type	Description
J1	DC5V0	DC power jack	5 V power connector
J6	CAN	1x3-pin header	HS CAN connector

Table 2. EVSE-SIG-BRD1X connectors*...continued*

Part identifier	PCB label	Connector type	Description
J9		3-position wire-to-board connector	Proximity pilot / control pilot connector
J15		SMA receptacle	Control pilot connector
J16		2x2-position receptacle	Control pilot connector
J23	GFCI	1x2-pin header	Secondary GFCI coil connector
J24	RELAY	2-position wire-to-board connector	Relay coil connector
J30	SWD DEBUG	2x5-pin header	LPC5536/LPC55S36 SWD debug connector
J33	SWD	9-pin (10-position) header	SJA1110B SWD debug connector
J34		2-position wire-to-board connector	100BASE-T1 Ethernet connector
J40		1x8-position receptacle	Arduino socket connectors
J36		1x10-position receptacle	
J37		1x8-position receptacle	
J39		1x6-position receptacle	
J44	EXP CN	2x20-pin header	Expansion connector
J45	MFP LIN/SPI	2x13-position receptacle	Multi-function port (MFP) connector
J46	UART	1x4-pin header	Auxiliary UART connector
JP1		1-pin header	Proximity pilot connector
P4	100B-TX	RJ45 jack	100BASE-TX Ethernet connector

1.5 Jumpers

Figure 6 shows the EVSE-SIG-BRD1X jumpers.

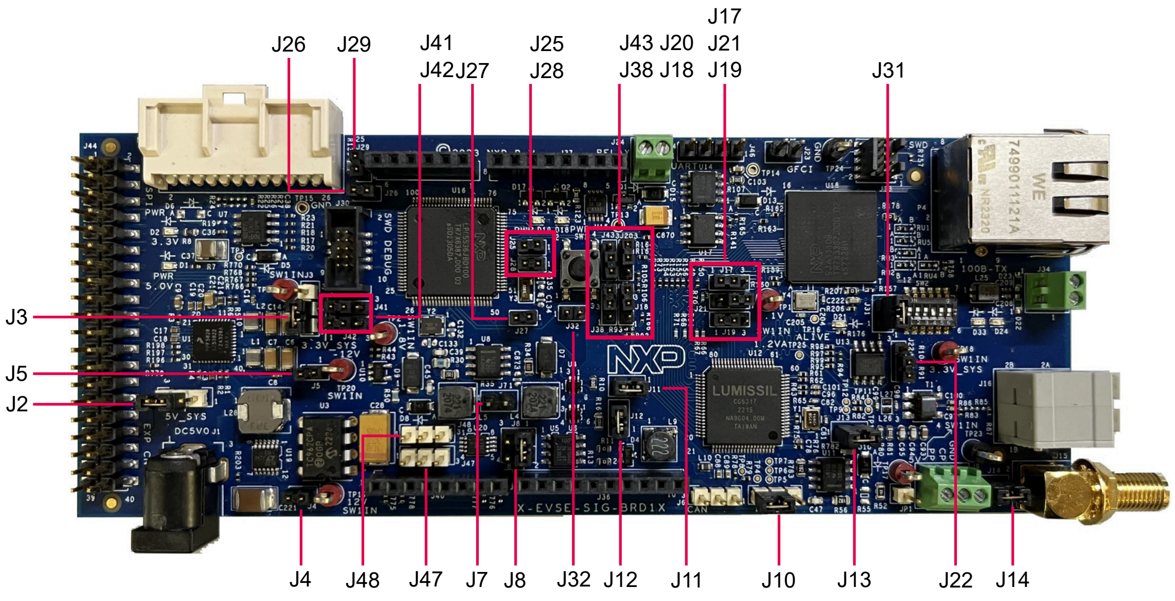

Figure 6. Jumpers

Table 3 describes the EVSE-SIG-BRD1X jumpers.

Table 3. EVSE-SIG-BRD1X jumpers

Part identifier	Jumper type	Description
J2	1x3-pin header	5V_SYS power source selection jumper: • Pins 1-2 shorted: 5V_SYS supply is produced from DC_5V_IN supply. • Pins 2-3 shorted (default setting): 5V_SYS supply is produced from 5V_ARD_EXP_CN supply.
J3	1x3-pin header	3.3V_SYS power source selection jumper: • Pins 1-2 shorted: 3.3V_SYS supply is produced from VDD_3V3 supply. • Pins 2-3 shorted (default setting): 3.3V_SYS supply is produced from 3V3_ARD_EXP_CN supply.
J4	1x2-pin header	12V0_ISO supply enable jumper: • Open: 12V0_ISO supply is OFF. • Shorted (default setting): 12V0_ISO supply is produced from 12V0 supply.
J5	1x2-pin header	-12V0_ISO supply enable jumper: • Open: -12V0_ISO supply is OFF. • Shorted (default setting): -12V0_ISO supply is produced from -12V0 supply.
J7	1x2-pin header	EVSE/EV PWM loopback enable jumper: • Open (default setting): EVSE/EV PWM loopback is disabled. • Shorted: EVSE/EV PWM loopback is enabled.
J8	2x2-pin header	Control pilot selection jumper: • Pins 1-2 shorted (default setting): EVSE control pilot is selected for PWM generation and detection. • Pins 3-4 shorted: EV control pilot is selected for PWM generation and detection.

Table 3. EVSE-SIG-BRD1X jumpers*...continued*

Part identifier	Jumper type	Description
J10	1x3-pin header	Proximity pilot control jumper: • Pins 1-2 shorted (default setting): Proximity pilot is used for EVSE simulation. • Pins 2-3 shorted: Proximity pilot is used for EV simulation.
J11	1x2-pin header	3V3_CG5317 supply enable jumper: • Open: 3V3_CG5317 supply is OFF. • Shorted (default setting): 3V3_CG5317 supply is produced from 3.3 V_SYS supply.
J12	1x2-pin header	VCORE supply enable jumper: • Open: VCORE supply is OFF. • Shorted (default setting): VCORE supply is produced from 3V3_ CG5317 supply.
J13	1x2-pin header	3.3VA supply enable jumper: • Open: 3.3VA supply is OFF. • Shorted (default setting): 3.3VA supply is produced from 3.3V_SYS supply.
J14	1x2-pin header	EVSE/EV control pilot I/O control jumper: • Open: EVSE/EV control pilot is disconnected from J15/J16. • Shorted (default setting): EVSE/EV control pilot is connected to J15/ J16.
J17	1x3-pin header	HPGP (CG5317) bootstrap pin control jumpers. For more details, see Table 10.
J18	1x3-pin header	
J19	1x3-pin header	
J20	1x3-pin header	
J21	1x3-pin header	
J22	1x3-pin header	
J25	1x2-pin header	MCU_VDD supply enable jumper: • Open: MCU_VDD supply is OFF. • Shorted (default setting): MCU_VDD supply is produced from 3.3V_ SYS supply.
J26	1x2-pin header	MCU_VDDA supply enable jumper: • Open: MCU_VDDA supply is OFF. • Shorted (default setting): MCU_VDDA supply is produced from 3.3 V_SYS supply.
J27	1x2-pin header	MCU_MAIN supply enable jumper: • Open: MCU_MAIN supply is OFF. • Shorted (default setting): MCU_MAIN supply is produced from 3.3 V_SYS supply.
J28	1x2-pin header	MCU_VBAT supply enable jumper: • Open: MCU_VBAT supply is OFF. • Shorted (default setting): MCU_VBAT supply is produced from 3.3 V_SYS supply.
J29	1x2-pin header	LPC5536/LPC55S36 MCU boot mode selection jumper: • Open: LPC5536/LPC55S36 MCU boots in In-System Programming (ISP) mode.

Table 3. EVSE-SIG-BRD1X jumpers*...continued*

Part identifier	Jumper type	Description
		• Shorted (default setting): LPC5536/LPC55S36 MCU boots in Normal mode (from internal flash memory).
J31	1x2-pin header	VCC_3V3_S supply enable jumper: • Open: VCC_3V3_S supply is OFF. • Shorted (default setting): VCC_3V3_S supply is produced from 3.3 V_SYS supply.
J32	1x2-pin header	SJA1110B SPI host connection enable jumper: • Open (default setting): SJA1110B SPI interface cannot connect to a host controller board. • Shorted: SJA1110B SPI interface (master) can connect to a host controller board (slave).
J38	1x3-pin header	HPGP (CG5317) SPI master selection jumper: • Pins 1-2 shorted (default setting): Host controller SPI chip select 1 is connected. • Pins 2-3 shorted: Host controller SPI chip select 2 is connected.
J41	1x3-pin header	Arduino socket connector J40 UART port control jumpers: • Pins 1-2-3 open: J40 UART port is connected to the expansion connector J44 UART port. • Pins 1-2 shorted (default setting): J40 UART port is connected to the LPC5536/LPC55S36 UART port. • Pins 2-3 shorted: J40 UART port is connected to the HPGP (CG5317 PHY) UART port.
J42	1x3-pin header	
J43	1x3-pin header	Host controller SPI interrupt source selection jumper: • Pins 1-2 shorted (default setting): HPGP (CG5317) SPI interface is selected as the interrupt source. • Pins 2-3 shorted: SJA1110B switch is selected as the interrupt source.
J47	1x3-pin header	EV charging ventilation option selection jumper: • Pins 1-2 shorted (default setting): 1.3 kΩ resistance (R47) is ON, the vehicle can be charged in an unventilated area. • Pins 2-3 shorted: 530 Ω resistance (R46 + R772) is ON, the vehicle can be charged only in a ventilated area. ***Note:*** *Both J47 and J48 serve the same purpose. J47 is used for MCU-controlled switching whereas J48 is used for manual switching. Only one of them can be used at a time. By default, J47 is used.*
J48	1x3-pin header	

1.6 Push button and DIP switch

EVSE-SIG-BRD1X has one push button SW1 and one dual inline package (DIP) switch SW2, as shown in Figure 7.

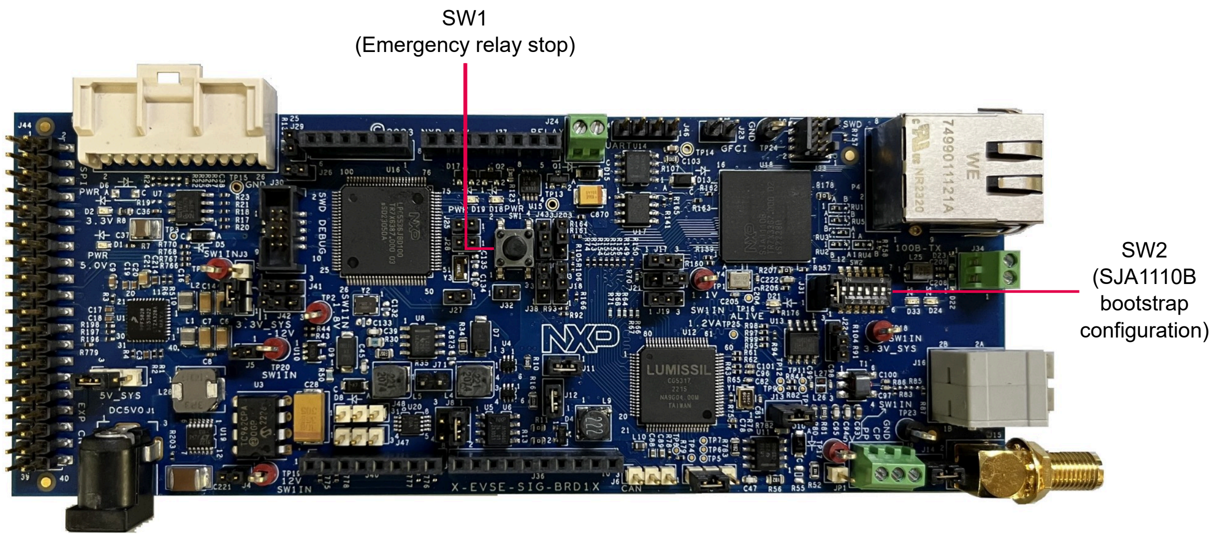

Figure 7. Push button and DIP switch

Table 4 describes the EVSE-SIG-BRD1X push button.

Table 4. EVSE-SIG-BRD1X push button

Part identifier	Supported function	Description
SW1	Emergency relay stop button	This push button can be used to turn OFF the relay during an emergency. Usually, the LPC5536/LPC55S36 MCU is used to turn ON / turn OFF the relay. For more details, see Section 2.6.2.

SW2 is a 6-pin DIP switch that allows to control the power-on bootstrap functions for the SJA1110B Ethernet switch on EVSE-SIG-BRD1X.

Each pin of the DIP switch has two positions:

- OFF position (pin has value 0)
- ON position (pin has value 1)

A DIP switch pin can be moved manually from OFF position to ON position and vice versa.

Table 5 describes the SW2 settings / SJA1110B bootstrap configuration.

Table 5. SW2 settings / SJA1110B bootstrap configuration

SW2 pins	Supported function	Description
SW2[1]	One-time-programmable (OTP)	• 1: Always ON position (default setting)
SW2[2:3]	SJA1110B boot mode selection	BOOT_OPTION[0:1]: • 00: Serial Download mode. An image is downloaded at Linux boot time. • 01: Boot from EEPROM (reserved) • 10: Boot from SPI flash • 11: Boot from QSPI flash (default setting)
SW2[4]	PHY master/slave selection	PHY_M_S5: • 0: PHY slave port (default setting) • 1: PHY master port

Table 5. SW2 settings / SJA1110B bootstrap configuration*...continued*

SW2 pins	Supported function	Description
SW2[5]	PHY automatic polarity detection	PHY_AUTO_POL_DET: • 0: If polarity is wrong, link training is blocked. • 1: Fully automated polarity detection and correction for 100 BASE-T1 PHY port 5 (default setting)
SW2[6]	PHY automatic mode selection	PHY_AUTO_MODE: Automatic mode select: • 0: Managed mode • 1: Automatic mode. The 100BASE-T1 PHY starts link training automatically. (default setting)

1.7 LEDs

EVSE-SIG-BRD1X provides numerous light-emitting diodes (LEDs) for monitoring system status. The information collected from the LEDs can be used for debugging purposes.

Figure 8 shows the EVSE-SIG-BRD1X LEDs.

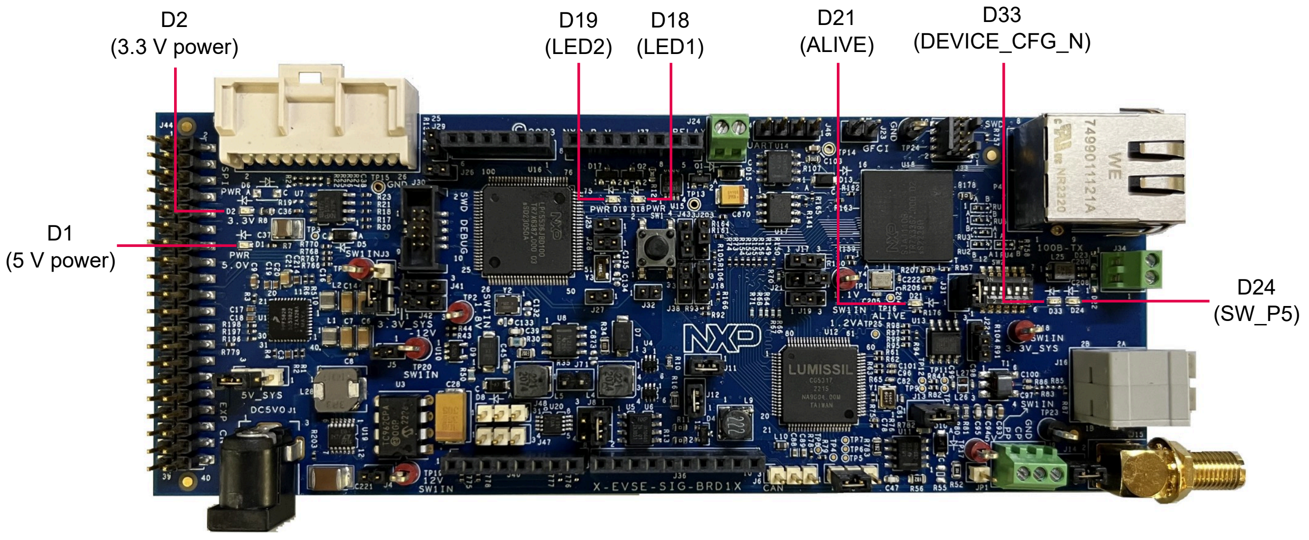

Figure 8. LEDs

Table 6 describes the EVSE-SIG-BRD1X LEDs.

Table 6. EVSE-SIG-BRD1X LEDs

Part identifier	PCB label	LED color	Description (when LED is ON)
D1	PWR 5.0V	Green	5V_SYS supply is available.
D2	PWR 3.3V	Green	3.3V_SYS supply is available.
D18	PWR	Green	User application LED1. According to the software implementation, the D18 LED function is defined as follows: • EV mode: The D18 LED blinks continuously. • EVSE mode: The D18 LED indicates the ground fault circuit interrupter (GFCI) fault status as follows: – No fault detected: The LED remains OFF. – Fault detected: The LED blinks continuously.
D19	PWR	Green	User application LED2. According to the software implementation, the D19 LED function is defined as follows:

Table 6. EVSE-SIG-BRD1X LEDs*...continued*

Part identifier	PCB label	LED color	Description (when LED is ON)
			• EV mode: The D19 LED blinks continuously. • EVSE mode: The D19 LED blinks continuously. If the LPC5536/ LPC55S36 device enters into Deep-Sleep mode, the D19 LED turns OFF to save power.
D21	ALIVE	Green	SJA1110B is up and running.
D24		Green	Link activity is in progress for SJA1110B switch 100BASE-T1 port 5.
D33		Green	SJA1110B switch subsystem configuration is complete.

2 Functional description

This section contains the following subsections:

- Section 2.1 "Power supplies"
- Section 2.2 "Clocks"
- Section 2.3 "Proximity pilot"
- Section 2.4 "Control pilot"
- Section 2.5 "GFCI circuit"
- Section 2.6 "Relay driver circuit"
- Section 2.7 "LPC5536/LPC55S36 MCU"
- Section 2.8 "SJA1110B switch"
- Section 2.9 "UART interface"
- Section 2.10 "Host notification"
- Section 2.11 "Meter notification"
- Section 2.12 "Power management with Deep-Sleep mode"
- Section 2.13 "CAN PHY"
- Section 2.14 "LIN PHY"
- Section 2.15 "Host connectors"

2.1 Power supplies

EVSE-SIG-BRD1X draws power from the host EVK connectors, for example, Arduino, EXP CN, or MFP. Within the board:

- A boost converter is used to generate a +12 V supply, which is used for PWM signaling of the control pilot and for driving the relay driver.
- A charge pump inverter is used to generate a -12 V supply, which is used for control pilot PWM signaling.

The board also has a DC power jack for supplying 5 V external power. The 5 V power can be used to drive external relays (typically 140 mA or above), if sufficient power is not drawn from the host controller board. The SJA1110B Ethernet switch on the board requires 1.1 V for its core operation. A power management integrated circuit (NXP PPF5020 PMIC) is used to generate the 3.3 V and 1.1 V power supplies from a 5 V power source.

2.1.1 Block diagram

Figure 9 shows the EVSE-SIG-BRD1X power supply diagram. It shows how power is supplied to different hardware components of the board.

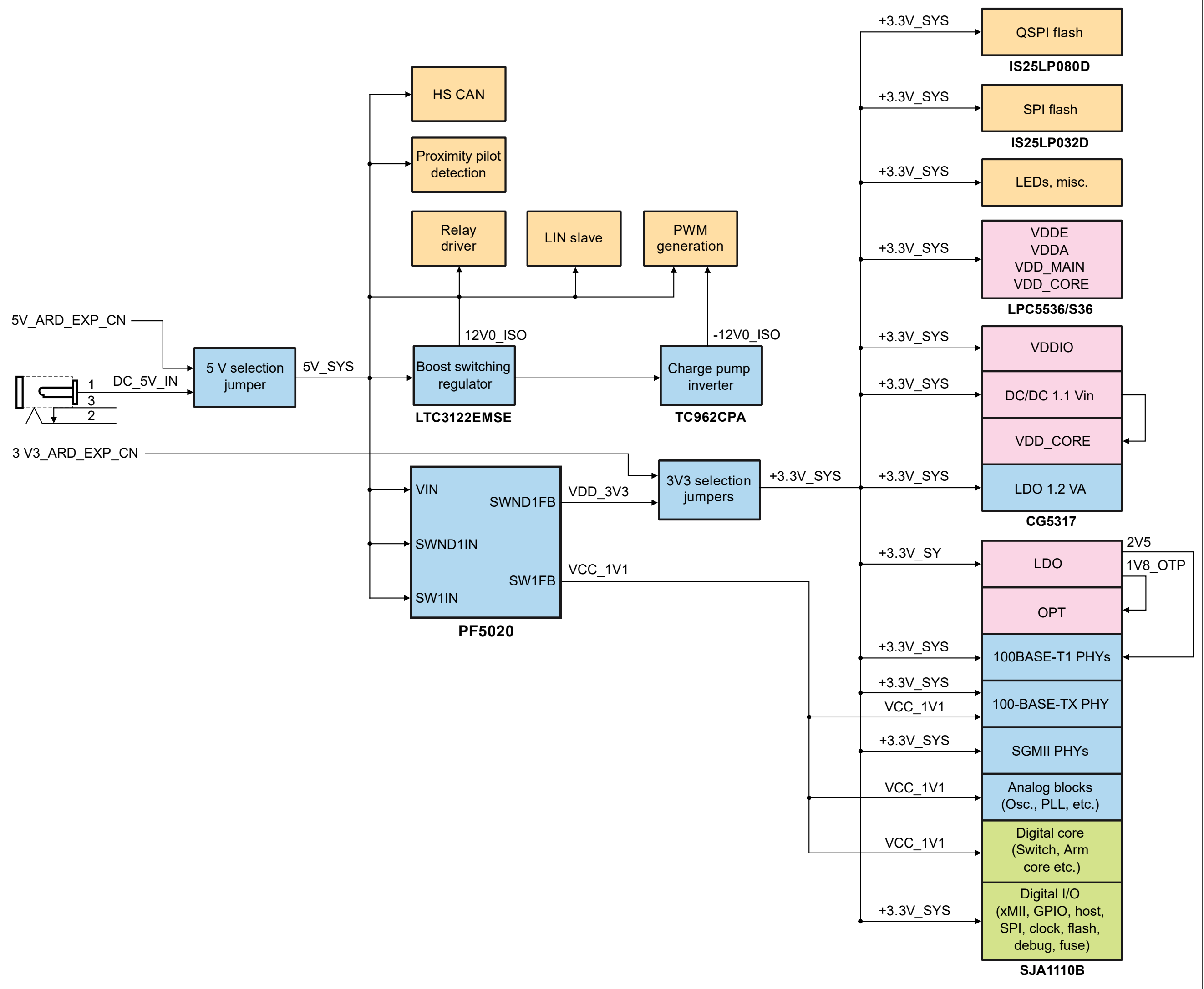

Figure 9. Power supply block diagram

2.1.2 Power supply sources

Table 7 shows the power supply sources available on EVSE-SIG-BRD1X.

Table 7. Power supply sources

Power source	Manufacturing part number	Power supply rail	Description
Through J37 pin 5		5V_ARD_EXP_CN	Power supply received from host board with Arduino, EXP CN, or multi-function port (S32G-VNP-RDB2)
Through J44 pins 2 and 4			
Through J45 pin 1			
Through DC power jack J1		DC_5V_IN	Power supply received from an external 5 V DC power source
Through J2 pin 2		5V_SYS	Board 5 V power. It can be either selected from 5V_ARD_EXP_CN or DC_5V_IN or supplied from an external power source.

Table 7. Power supply sources*...continued*

Power source	Manufacturing part number	Power supply rail	Description
Through J37 pin 4		3V3_ARD_EXP_CN	Power supply received from host board with Arduino, EXP CN, or multi-function port (S32G-VNP-RDB2)
Through J44 pins 1 and 17			
Through J45 pin 3			
U1	PPF5020CMMAYES	VDD_3V3	3.3 V output from PMIC
Through J3 pin 2		3.3V_SYS	Board 3.3 V power. It can be either selected from 3V3_ARD_EXP_CN or VDD_3V3 or supplied from an external power source.
U1	PPF5020CMMAYES	VCC_1V1	1.1 V output from PMIC that is the power supply for the SJA1110B switch core.
U19	LTC3122EMSE#PBF	12V0	Board 12 V power that powers the control pilot PWM op-amp and the relay MOSFET.
U3	TC962CPA	-12V0	Board 12 V power that powers the control pilot PWM op-amp.

2.1.3 Schematic design

The input power supplies of EVSE-SIG-BRD1X are 5 V and 3.3 V. The PMIC PPF5020 also generates 3.3 V power. 3.3 V power can also be drawn from the host controller board through host connectors.

Table 8 shows the power requirements of the board, along with maximum values. The typical requirements should be much less.

Table 8. Power consumption calculation

Block	Current specifications for power supplies		
	3.3 V	1.1 V	+12 V, -12 V
LPC5536/LPC55S36	15 mA		
CG5317	226 mA		
SJA1110B	228 mA typical, 358 mA max	183 mA typical, 1095 mA max	
Control pilot circuit			48 mA, 48 mA
Two relays			280 mA
Total current	>599 mA	1095 mA max	>376 mA
Total power	1.98 W	1.2 W	4.52 W
Sum of total powers	1.98 W + 1.2 W + 4.52 W = 7.7 W		

The following are some design considerations related to power:

- 3.3 V supply rail from Arduino connector / EXP CN / MFP connector / PMIC PPF5020 drives > 600 mA load current.
- 5 V supply rail from Arduino connector / EXP CN / MFP connector / external power source is required for board load requirements.
- 1.1 V supply rail from the PMIC drives > 1.1 A load current.

- +12 V supply rail from boost converter drives up to 800 mA load current. It also supplies +12 V power to the charge pump inverter.
- -12 V supply rail from charge pump inverter drives up to 80 mA load current.

Although in most use cases, the 5 V and 3.3 V power supplies drawn from the host connectors meet the power requirements of the board. However, the board can also be powered from an external power source through the DC power jack J1. The power received through J1 can be used to drive the relays.

2.1.3.1 PMIC PPF5020

The power management integrated circuit (PMIC) PF5020 integrates multiple high-performance buck regulators. It has a built-in one-time programmable (OTP) memory for storing the startup configurations. The OTP memory drastically reduces the external components that are typically used to set the output voltage and sequence of regulators.

The following are some important features of the PMIC PPF5020:

- Three buck converters with high efficiency
- One linear regulator with load switch options
- Real-time clock (RTC) supply and coin cell charger
- Watchdog timer/monitor
- One-time programmable device configuration
- 3.4 MHz I^{2}C communication interface
- 40-pin quad flat no-lead (QFN) package with wettable flank and exposed pad

Figure 10 shows the schematic design using PMIC PPF5020.

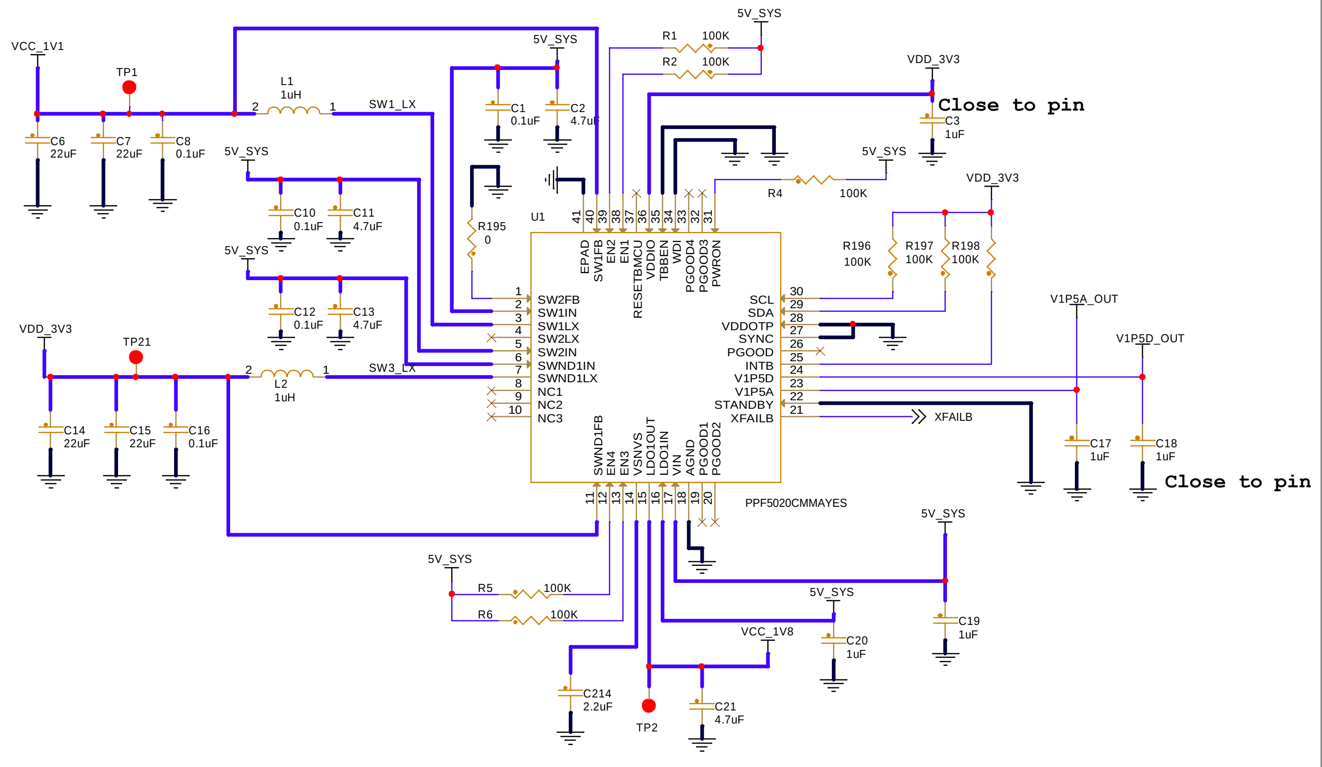

Figure 10. PMIC circuit design

The OTP memory of PPF5020 is programmed to generate the 1.1 V supply before the 3.3 V supply. The 1.1 V supply is connected to switch SJA1110B. Figure 11 shows the power-on sequence and timing of PPF5020.

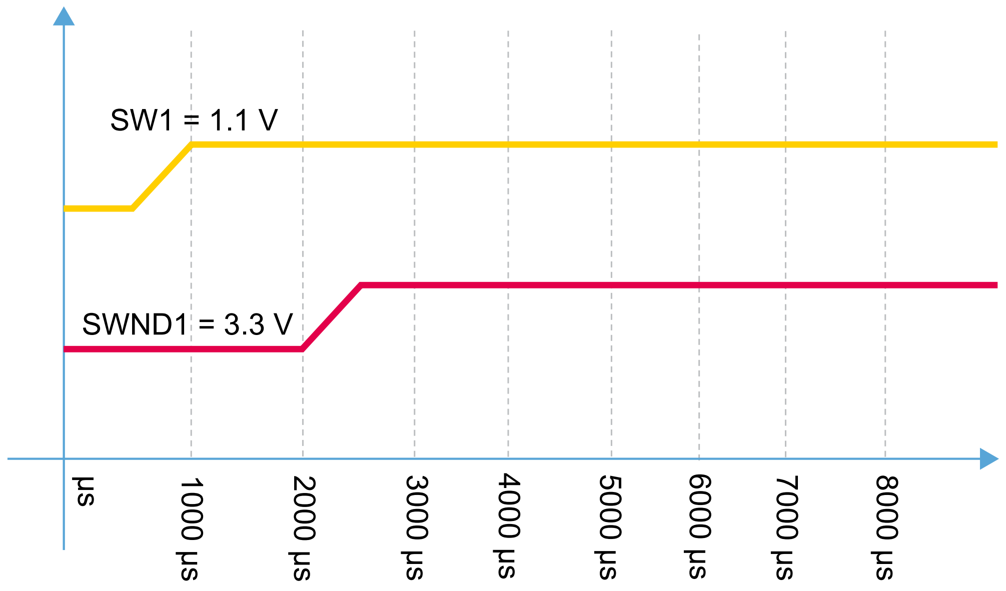

Figure 11. PMIC PPF5020 power sequence

Note: *To achieve the PMIC PPF5020 power sequence shown in Figure 11, use the 3.3 V supply from the PMIC, instead of the 3.3 V supply from the host connectors.*

2.1.3.2 Boost converter and charge pump inverter

EVSE-SIG-BRD1X has an LTC3122 step-up DC-DC converter, which acts as a boost converter. The LTC3122 converter is configured in the boost converter hardware configuration. It can drive a current of up to 800 mA for V_{IN} = 5 V and V_{OUT} = 12 V.

Figure 12 shows the boost converter schematic design.

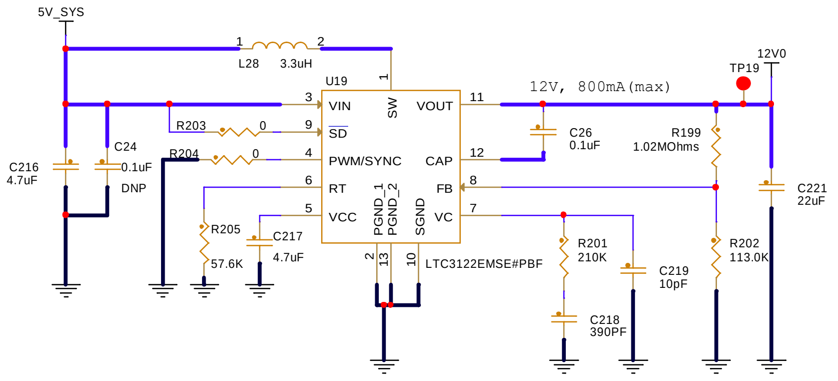

Figure 12. Boost converter design

The charge pump inverter is designed with TC962CPA and it generates -12 V from the +12 V DC input.

Figure 13 shows the charge pump inverter schematic design.

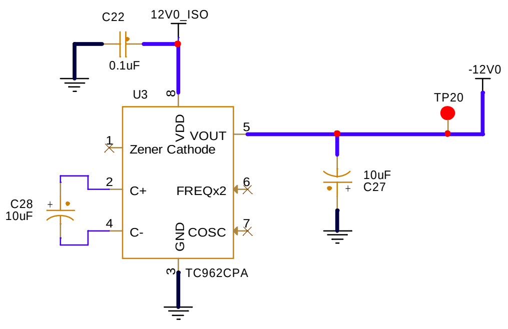

Figure 13. Charge pump inverter design

2.2 Clocks

Table 9 describes the clocks available on EVSE-SIG-BRD1X.

Table 9. EVSE-SIG-BRD1X clocks

Clock generator	Clock	Frequency	Destination
Crystal Y2	XTAL32M_[P,N]	16 MHz	LPC5536/LPC55S36 MCU
Crystal Y3	XTAL32K_[P,N]	32.768 kHz	
Crystal Y1	XTI, XTO	25 MHz	CG5317
Crystal Y4	OCI_[IN,OUT]	25 MHz	SJA1110B

2.3 Proximity pilot

The proximity detection scheme is shown in Figure 14.

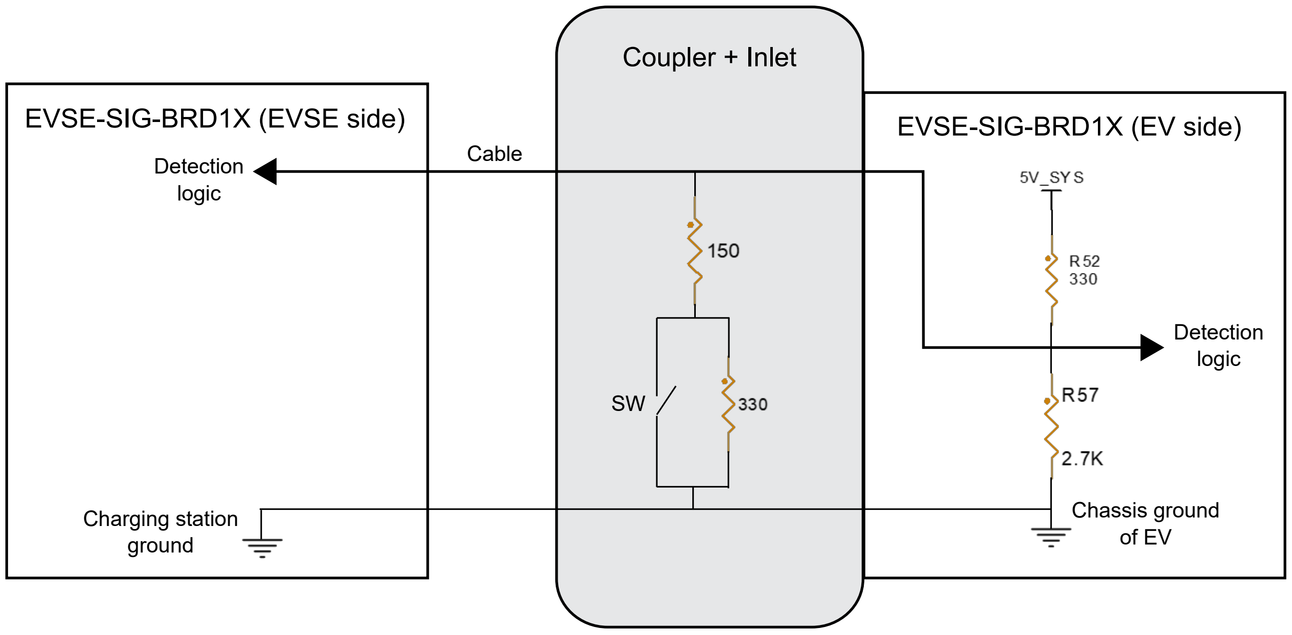

Figure 14. Proximity pilot detection logic

When the vehicle coupler connector is not connected to the vehicle inlet, the onboard R52 and R57 resistor divider produces a voltage level of 4.46 V. This node is fed to a detection logic inside the EVSE-SIG-BRD1X vehicle side simulation. When the coupler is connected to the inlet and the latch release actuator switch is closed, the voltage at the point of detection logic drops to 1.53 V. When the latch switch is open, the detection voltage is changed to 2.77 V.

2.3.1 Schematic design

The proximity pilot circuit includes level sensing of the connector signal to trigger a wake-up to the EVSE or EV. The level of the proximity pilot signal is measured to determine the current state of proximity detection, as shown in Figure 15.

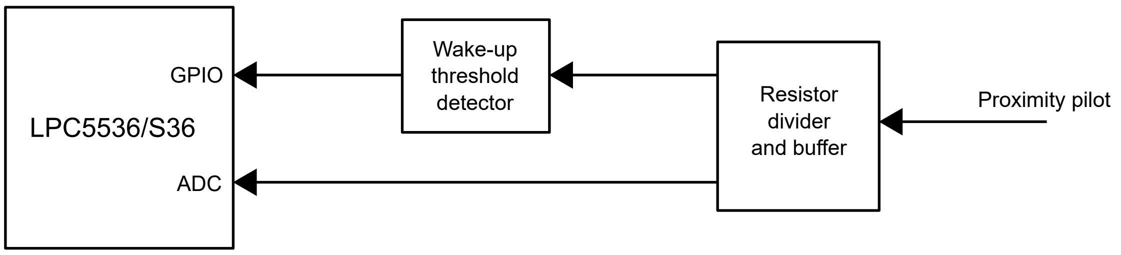

Figure 15. Proximity detection and wake-up

Figure 16 shows proximity detection circuit schematic design.

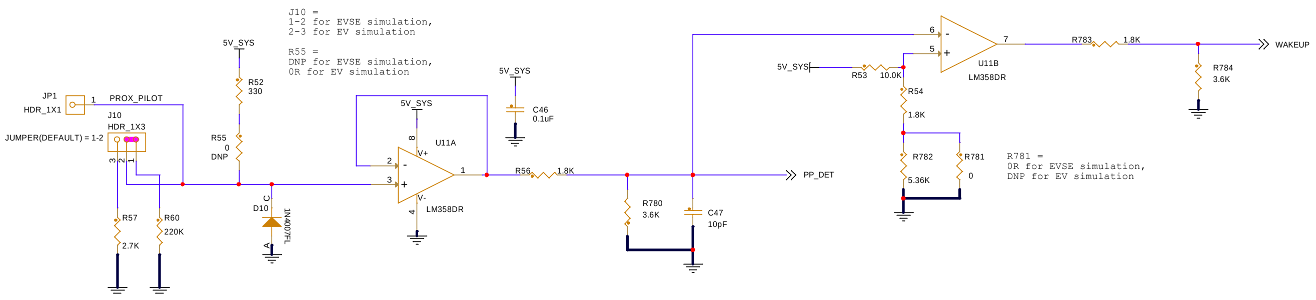

Figure 16. Proximity detection design

To simulate the **EV side** of the detection logic:

- J10 pins 2 and 3 are shorted.
- R55 is populated.
- R781 is removed.

R52 and R57 together create the input resistor network for the detection logic, which starts with PROX_PILOT and is buffered at op-amp U11A. The +5 V op-amp output is reduced to 3.3 V level to interface with the onboard LPC5536/LPC55S36 MCU through PP_DET. PP_DET is also compared at U11B with a threshold set at 2.1 V:

- If the coupler is not connected to the inlet, PP_DET = 2.4 V and the WAKEUP signal becomes low.
- If the coupler is connected and the switch in Figure 14 is closed, PP_DET = 1 V and the WAKEUP signal becomes high.
- If the coupler is connected and the switch in Figure 14 is open, PP_DET = 1.8 V and the WAKEUP signal becomes high.

The LPC5536/LPC55S36 ADC can determine the individual levels precisely by measuring PP_DET.

Note: *The selected op-amp LM358DR has a rail-to-output drop of approximately 1.4 V. Therefore, the voltage levels at PP_DET and the threshold of U11B negative input must be calculated accordingly.*

To simulate the **EVSE side** of the detection logic:

- J10 pins 1 and 2 are shorted.
- R55 is removed (default setting).
- R781 is populated (default setting).

R60 provides a weak pull-down resistor to the detection logic PROX_PILOT. Therefore, when the cable coupler is not connected to the vehicle inlet, U11A output is low. In this case, the threshold voltage of U11B positive input is set to 0.76 V.

- If the coupler is not connected to the inlet, PP_DET = 0 V and the WAKEUP signal becomes high.
- If the coupler is connected and the switch is closed, PP_DET = 1 V and the WAKEUP signal becomes low.
- If the coupler is connected and the switch is open, PP_DET = 1.8 V and the WAKEUP signal becomes low.

2.3.2 Software implementation

For the PROX_PILOT signal detection in the EVSE side of EVSE-SIG-BRD1X, the WAKEUP signal is configured with GPIO input pin configuration. The below code snippet is based on the NXP MCUXpresso SDK:

```
const uint32_t port0_pin28_config = (/* Pin is configured as PIO0_28 */
          IOCON_PIO_FUNC0 |
    /* Selects pull-up function */
    IOCON_PIO_MODE_PULLUP |
    /* Standard mode, output slew rate control is enabled */
    IOCON_PIO_SLEW_STANDARD |
    /* Input function is not inverted */
    IOCON_PIO_INV_DI |
    /* Enables digital function */
    IOCON_PIO_DIGITAL_EN |
    /* Open drain is disabled */
    IOCON_PIO_OPENDRAIN_DI |
    /* Analog switch 0 is open (disabled) */
    IOCON_PIO_ASW0_DI);
        /* GFCI_INT: PORT0 PIN28 (coords: 66) is configured as PIO0_28 */
        IOCON_PinMuxSet(IOCON, 0U, 28U, port0_pin28_config);
        gpio_pin_config_t WAKEUP_config = {
          .pinDirection = kGPIO_DigitalInput,
    .outputLogic = 0U
        };
        /* Initialize GPIO functionality on pin PIO0_28 (pin 66)  */
        GPIO_PinInit(GPIO, 0U, 28U, &WAKEUP_config);
```

The `PP_Process()` function reads the WAKEUP signal state periodically through polling:

```
ppValRead = GPIO_PinRead(GPIO, BOARD_PP_WAKEUP_PORT, BOARD_PP_WAKEUP_PIN);
```

Alternatively, a GPIO interrupt from the PINT module of the LPC5536/LPC55S36 MCU can read the WAKEUP signal state.

PP_DET is configured as an ADC input channel:

```
const uint32_t port1_pin19_config = (/* Pin is configured as ADC0_4B */
          IOCON_PIO_FUNC0 |
       /* No addition pin function */
       IOCON_PIO_MODE_INACT |
       /* Standard mode, output slew rate control is enabled */
       IOCON_PIO_SLEW_STANDARD |
       /* Input function is not inverted */
```

```
        IOCON_PIO_INV_DI |
        /* Enables analog function */
        IOCON_PIO_ANALOG_EN |
        /* Open drain is disabled */
        IOCON_PIO_OPENDRAIN_DI |
        /* Analog switch 0 is closed (enabled) */
        IOCON_PIO_ASW0_EN);
         /* PORT1 PIN19 (coords: 83) is configured as ADC0_4B */
         IOCON_PinMuxSet(IOCON, 1U, 19U, port1_pin19_config);
```

The ADC configuration is described in Section 2.4.1.7.

The ADC samples for PP_DET are read in the ADC interrupt handler function:

```
void DEMO_LPADC_IRQ_HANDLER_FUNC(void)
{
 g_LpadcInterruptCounter++;
#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT ==
 2U))
 if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcChnCPResultConfigStruct, 0U))
#else
  if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcChnCPResultConfigStruct))
#endif /* FSL_FEATURE_LPADC_FIFO_COUNT */
  {
   g_LpadcChnCPConversionCompletedFlag = true;
  }

#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT ==
 2U))
 if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcChnPPResultConfigStruct, 1U))
#else
  if (LPADC_GetConvResult(DEMO_LPADC_BASE, &g_LpadcChnPPResultConfigStruct))
#endif /* FSL_FEATURE_LPADC_FIFO_COUNT */
  {
   g_LpadcChnPPConversionCompletedFlag = true;
  }

 SDK_ISR_EXIT_BARRIER;
}
```

2.4 Control pilot

The control pilot of EVSE-SIG-BRD1X performs the following functions:

- Generates the J1772 PWM (IEC 61851) signal.
- Amplifies the signal to ±12 V.
- Detects if the signal level changes due to:
  - The connection of the charging cable to the EV.
  - The internal switching of the charging states.

An equivalent circuit is available at the EV side that:

- Measures the PWM ON time.
- Changes the switching levels to request the start/stop of charging sequence to the EVSE.

The 1 kHz PWM signal provides basic signaling between the EVSE and EV to indicate EV connect states, charging current capacity, and the charge start/stop requests. The PWM ON period can vary from 10% to 96% for basic signaling. PWM ON period of 3% to 7% is reserved to indicate high-level signaling using HPGP.

Figure 17 shows EVSE control pilot PWM.

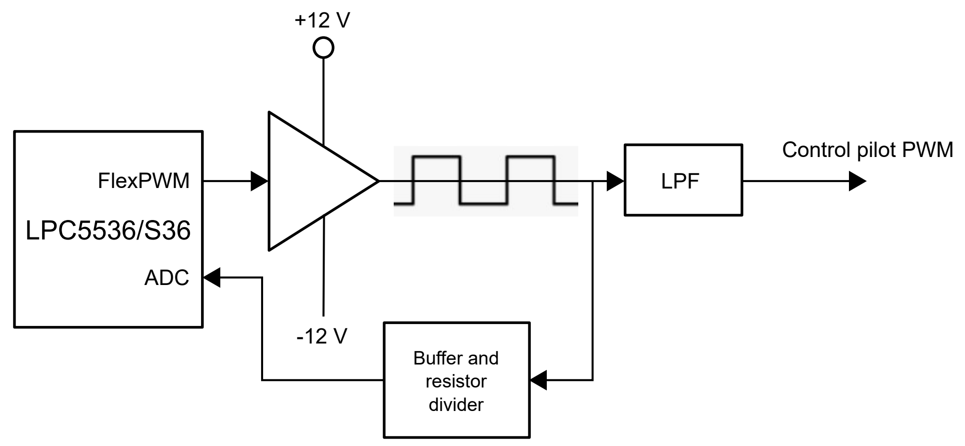

Figure 17. EVSE control pilot PWM

Figure 18 shows EV control pilot PWM.

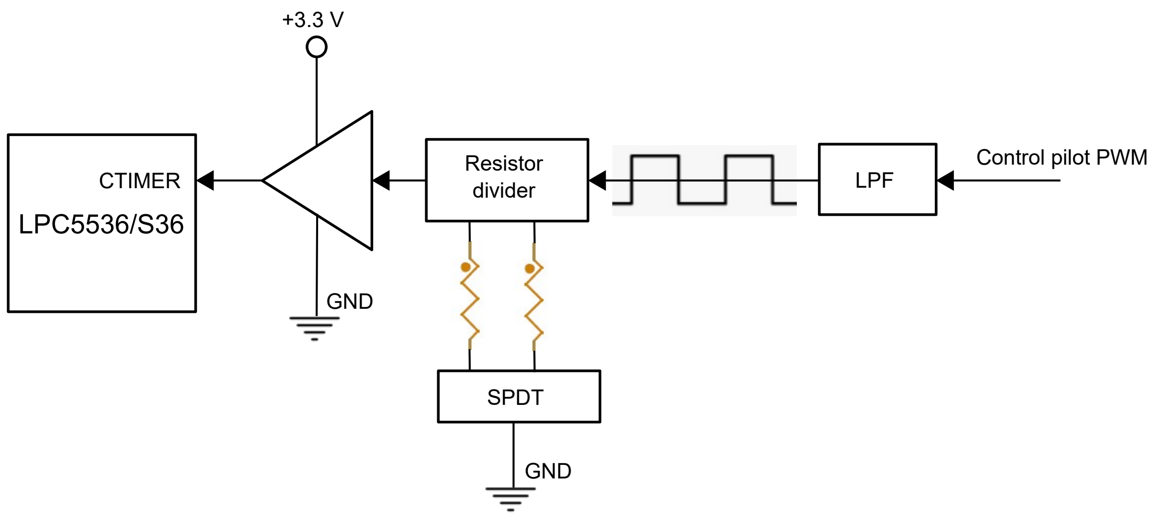

Figure 18. EV control pilot PWM

Note: *EVSE-SIG-BRD1X generates +12 V and -12 V from the 5 V input power sources.*

HomePlug Green PHY (HPGP)

The HPGP transceiver provides high-level signaling over the same control pilot signal of the charging cable. That is, the HPGP transceiver is superimposed onto the 5% J1772 PWM.

The HPGP transceiver supports:

- Signal Level Attenuation Characterization (SLAC)
- SECC Discovery Protocol (SDP)
- TCP/IP setup
- ISO 15118 communication sequences using orthogonal frequency division multiplexing (OFDM) with a carrier frequency of 2 to 30 MHz.
- A data rate of up to 10 Mbit/s using the communication link between the EVSE and EV.

The host processor can access the HPGP in EVSE-SIG-BRD1X using the SPI or Ethernet interface. These interfaces allow the host to:

- Boot
- Communicate
- Run control and management services with the HPGP

The SPI interface of EVSE-SIG-BRD1X can be accessed through:

- Arduino socket
- EXP CN connector
- MFP connector, which provides a seamless connection with an S32G-VNP-RDB3 board.

The host controller can also interface with the onboard HPGP through a 100BASE-T1 or 100BASE-TX port.

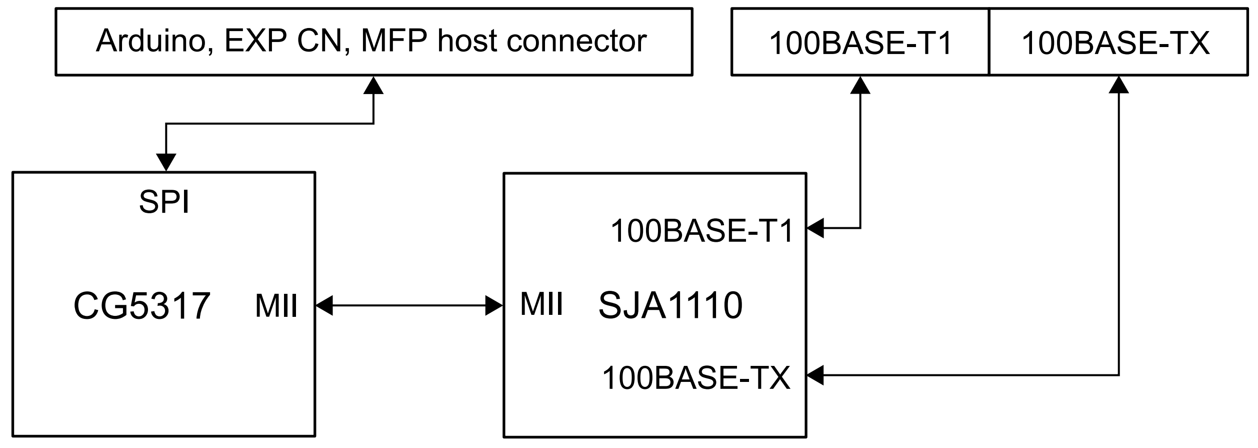

Figure 19. HPGP host interfacing

2.4.1 J1772 PWM

The control pilot circuit in EVSE-SIG-BRD1X includes both the generation (EVSE side) and sense (EVSE and EV sides) capabilities. It is implemented to support both these capabilities for EVSE and vehicle interfaces.

Figure 20 shows the PWM generation and sense scheme on the EVSE and EV sides.

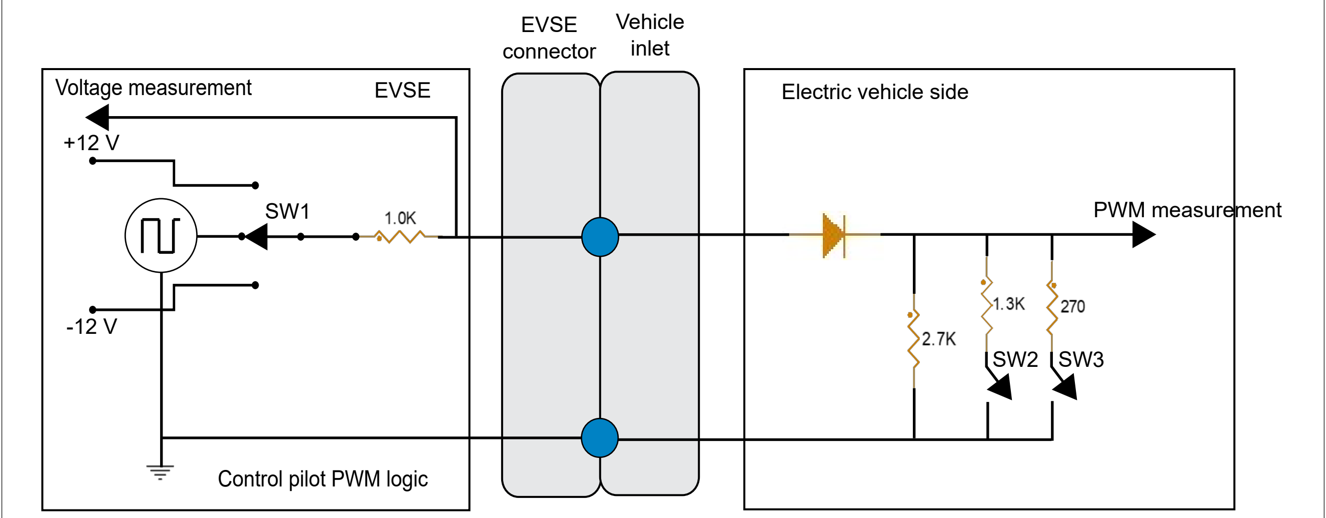

Figure 20. Control pilot PWM generation and sense

2.4.1.1 Schematic design

The J1772 PWM circuit in EVSE-SIG-BRD1X is configurable to the electric vehicle supply equipment (EVSE) side or electric vehicle (EV) side interface using board jumper settings. The implementation is shown in Figure 21.

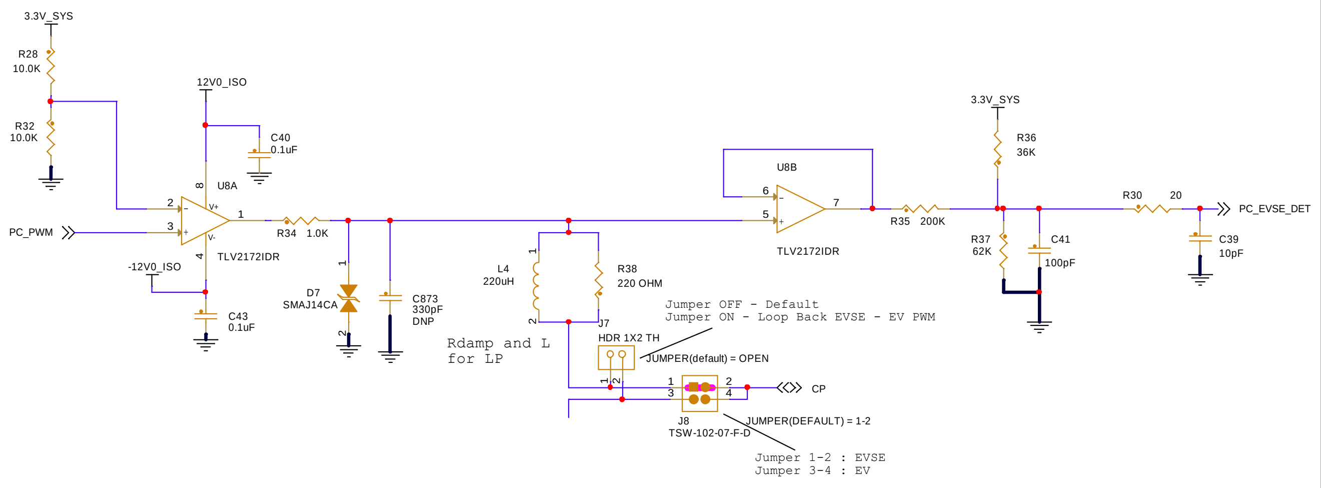

Figure 21. EVSE side circuit of control pilot PWM

The control pilot EVSE circuit shown in Figure 21 amplifies the PC_PWM signal, which is a 1 kHz PWM signal generated using the LPC5536/LPC55S36 MCU. U8 is a high-speed rail-to-rail op-amp with fast rise time to meet the requirements of EVSE standards. The op-amp U8A is powered with the +12 V and -12 V supply rails and it produces the ±12 V PWM at its output. The output is passed through a low-pass-filter (L4 and R38) to filter out the high-frequency signal coupling occurred due to the high-level signaling produced by the HPGP CG5317 analog circuit. Therefore, the LPC5536/LPC55S36 MCU controls the levels of the PWM signal (CP) as follows:

- +12 V high level when the vehicle is not connected
- ±12 V PWM of 1 kHz
- -12 V indicating power fail or other errors at the EVSE side

Figure 22 shows the EV side circuit of control pilot PWM sense.

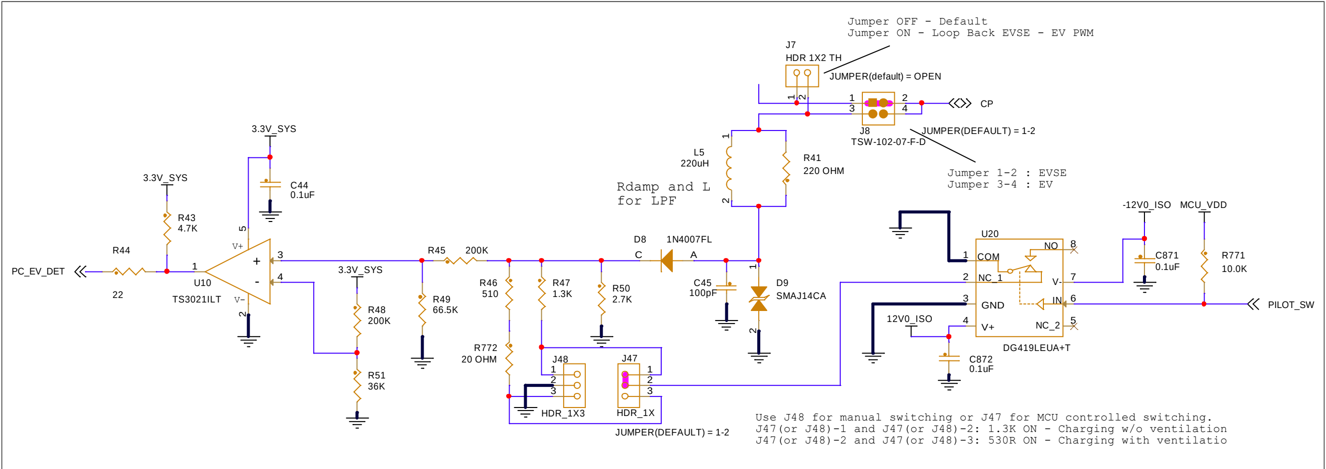

Figure 22. EV side circuit of control pilot PWM sense

The control pilot EV circuit shown in Figure 22 is only used for EV simulation. To simulate the electric vehicle side of EVSE-SIG-BRD1X, connect two EVSE-SIG-BRD1X boards to each other through a suitable charging cable control pilot wire.

After the incoming CP signal is decoupled from the high-level signaling component using a low-pass-filter (L5 and R41), the diode D8 forwards the positive half of the signal. The resistor network made from R50, R47, R46, R772, R45, and R49 divides the forwarded signal to the input range of 3.3 V or lower. The output of the op-amp U10 is sent to the LPC5536/LPC55S36 MCU to measure PWM voltage level, frequency, and ON time.

A basic communication sequence between the EVSE and EV is as follows:

1. Initially, if the EVSE can supply charge to the EV, it generates +12 V at the CP pin. It waits for a vehicle to get connected through the charging cable. This state is termed as state 'A'.
2. When an EV is connected, its resistance R50 (2.74 kΩ) drops the CP voltage level to approximately +9 V. The voltage drop is measured at both the EVSE and vehicle sides. The individual detection logic at each side can determine that EVSE and EV were connected just now. This state is termed as state 'B'.
3. Next, the vehicle can connect an internal switch (SW2 as in Section 2.4.1) to the resistance R47 (1.3 kΩ) in the circuit. It reduces the CP voltage level further to approximately +6 V. The EVSE detects this change in the voltage level and concludes that the EV is ready for the EVSE to start charging. Closing switch SW2 indicates that the vehicle can be charged in an unventilated area in the station. This state is termed as state 'C'.
4. Otherwise, the vehicle can connect the internal switch SW3 (as in Section 2.4.1) to an equivalent resistance of R47 || (R46 + R772). It gives a resistance value of approximately 372 Ω (see note below). Switching to this combination reduces the CP voltage level to approximately +3 V. The EVSE detects this change in the voltage level and concludes that the EV is ready for the EVSE to start charging. Closing switch SW3 indicates that the vehicle can be charged in a ventilated area in the station. This state is termed as state 'D'. It is not currently supported.
   Note: *Instead of using a resistor of the recommended resistance value 270 Ω, this resistance combination is used for the current design only for testing purposes. For actual design, use a 270 Ω resistor.*
   Another state, which is used for error conditions, is state 'E'. This state is also not currently supported by EVSE.
5. Now, the EVSE can start the PWM signal with the duty cycle ranging from typically 5% to 97%. 5% duty cycle indicates that the EVSE wants to inform the EV that it can also support high-level signaling using the HPGP CG5317. A higher value duty cycle indicates only the basic-level signaling, that is, the charge current rating of the EVSE. The electric vehicle side of EVSE-SIG-BRD1X measures this PWM frequency and duty cycle.

Note: *Refer to SAE J1772 standard (see Table 23) for details about the charge current encoded with PWM duty cycles.*

6. At the end of the charging, the EV can open SW2 or SW3. As a result, the CP voltage level comes back to +9 V, that is, state 'B'. On the EVSE side, EVSE-SIG-BRD1X monitors this voltage level continuously. Any change to +9 V indicates that the PWM can be stopped and the charging process can also be stopped from the EVSE side.
7. If the charging cable is disconnected, the voltage level of the CP pin automatically comes back to the +12 V level, that is, state 'A'.

2.4.1.2 eFlexPWM usage for control pilot for EVSE

In EVSE-SIG-BRD1X, the Enhanced Flexible Pulse Width Modulator (eFlexPWM) module of the LPC5536/ LPC55S36 MCU is used to generate J1772 PWM. eFlexPWM is also used internally for triggering the ADC at the correct position to determine precisely the level at the mid-point of the PWM ON period.

Figure 23 shows the ADC triggering timing diagram.

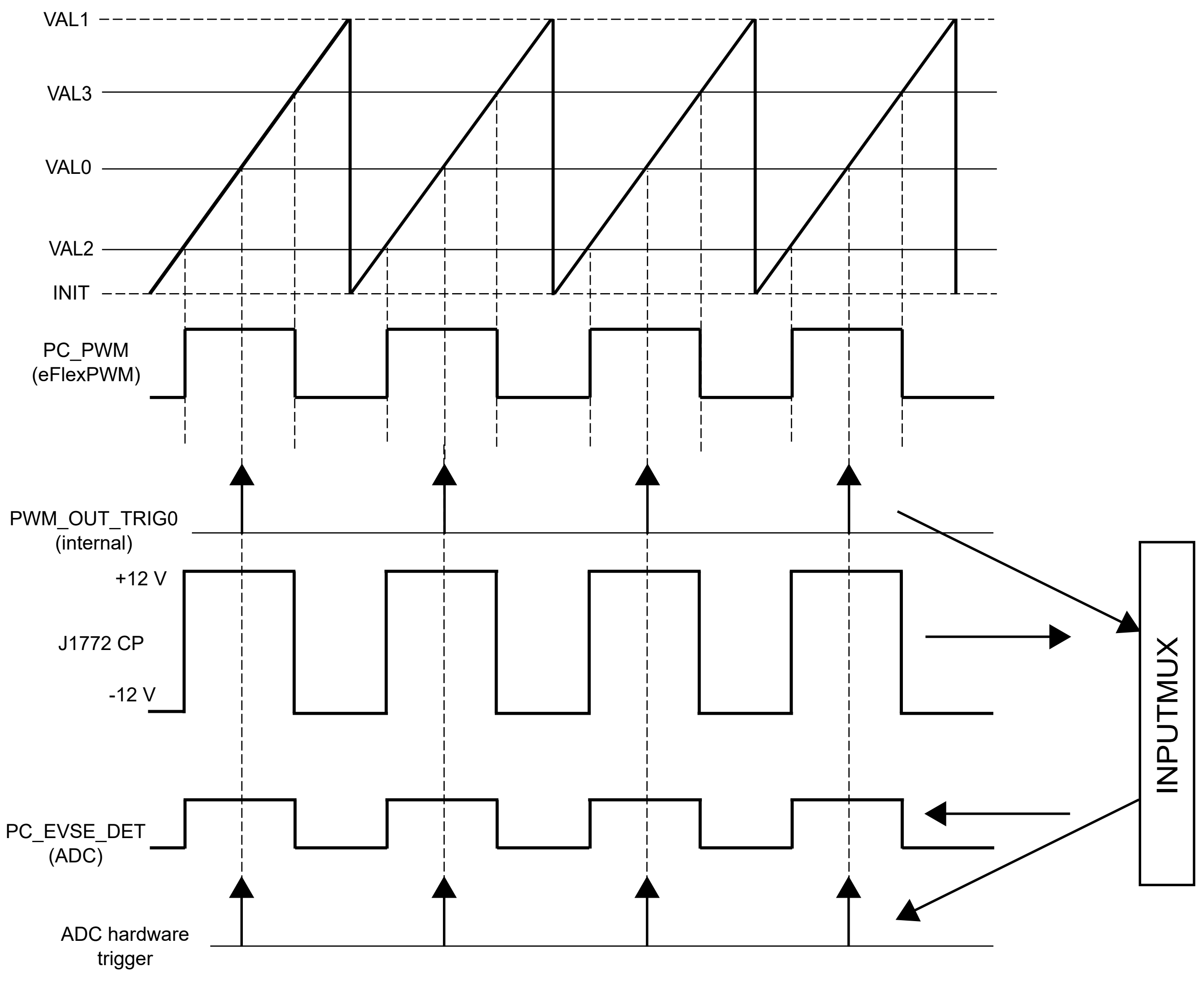

Figure 23. eFlexPWM and ADC triggering

To generate a center-aligned PWM of 1 kHz frequency, the INIT register of the eFlexPWM is loaded with half tick counts before zero value to initialize the PWM counter. When the PWM counter value reaches a negative VAL2, the PWM output changes to the high level.

Later, when the counter crosses VAL0 (= 0), the eFlexPWM module internally generates a trigger called PWM_OUT_TRIG0. Next, when the counter value equals VAL3, the output toggles back to the low level. When the counter value reaches VAL1, it is reinitialized to the INIT value to repeat the next PWM cycle.

To update the PWM ON time or duty cycle, VAL2 and VAL3 are updated.

The PWM1 module and its PWM_A channel generate:

- Control pilot PWM of fixed 1 kHz.
- A varying duty cycle of 3% to 97%.

The internal PWM_OUT_TRIG0 trigger is connected to the hardware trigger of the ADC channel internally within the LPC5536/LPC55S36 MCU. The internal cross-connection is implemented through the INPUTMUX module of the MCU.

2.4.1.3 Software implementation of eFlexPWM

eFlexPWM is initialized using the code below, which is based on the NXP MCUXpresso SDK:

```
void CP_Init(void)
{
    ctimer_config_t config;
    /*
    * pwmConfig.enableDebugMode = false;
    * pwmConfig.enableWait = false;
    * pwmConfig.reloadSelect = kPWM_LocalReload;
    * pwmConfig.clockSource = kPWM_BusClock;
    * pwmConfig.prescale = kPWM_Prescale_Divide_1;
    * pwmConfig.initializationControl = kPWM_Initialize_LocalSync;
    * pwmConfig.forceTrigger = kPWM_Force_Local;
    * pwmConfig.reloadFrequency = kPWM_LoadEveryOportunity;
    * pwmConfig.reloadLogic = kPWM_ReloadImmediate;
    * pwmConfig.pairOperation = kPWM_Independent;
    */
    PWM_GetDefaultConfig(&pwmConfig);
    #ifdef DEMO_PWM_CLOCK_DEVIDER
    pwmConfig.prescale = DEMO_PWM_CLOCK_DEVIDER;
    #endif
    /* Use full cycle reload */
    pwmConfig.reloadLogic = kPWM_ReloadPwmFullCycle;
    /* PWM A & PWM B form a complementary PWM pair */
    pwmConfig.pairOperation = kPWM_Independent;
    pwmConfig.enableDebugMode = true;
    pwmConfig.prescale = kPWM_Prescale_Divide_4;
    /* Initialize submodule 0 */
    if (PWM_Init(BOARD_PWM_BASEADDR, kPWM_Module_0, &pwmConfig) == kStatus_Fail)
    {
    PRINTF("PWM initialization failed\n");
    return;
    }
    /*
    * config->faultClearingMode = kPWM_Automatic;
    * config->faultLevel = false;
    * config->enableCombinationalPath = true;
    * config->recoverMode = kPWM_NoRecovery;
    */
    PWM_FaultDefaultConfig(&faultConfig);
    #ifdef DEMO_PWM_FAULT_LEVEL
    faultConfig.faultLevel = DEMO_PWM_FAULT_LEVEL;
    #endif
    /* Sets up the PWM fault protection */
    PWM_SetupFaults(BOARD_PWM_BASEADDR, kPWM_Fault_0, &faultConfig);
    PWM_SetupFaults(BOARD_PWM_BASEADDR, kPWM_Fault_1, &faultConfig);
    PWM_SetupFaults(BOARD_PWM_BASEADDR, kPWM_Fault_2, &faultConfig);
    PWM_SetupFaults(BOARD_PWM_BASEADDR, kPWM_Fault_3, &faultConfig);
    /* Set PWM fault disable mapping for submodule 0/1/2 */
    PWM_SetupFaultDisableMap(BOARD_PWM_BASEADDR, kPWM_Module_0, kPWM_PwmA,
 kPWM_faultchannel_0,
    kPWM_FaultDisable_0 | kPWM_FaultDisable_1 | kPWM_FaultDisable_2 |
 kPWM_FaultDisable_3);
    PWM_SetupFaultDisableMap(BOARD_PWM_BASEADDR, kPWM_Module_1, kPWM_PwmA,
 kPWM_faultchannel_0,
    kPWM_FaultDisable_0 | kPWM_FaultDisable_1 | kPWM_FaultDisable_2 |
 kPWM_FaultDisable_3);
    PWM_SetupFaultDisableMap(BOARD_PWM_BASEADDR, kPWM_Module_2, kPWM_PwmA,
 kPWM_faultchannel_0,
    kPWM_FaultDisable_0 | kPWM_FaultDisable_1 | kPWM_FaultDisable_2 |
 kPWM_FaultDisable_3);
    /* Enables the clock for the GPIO1 module */
    CLOCK_EnableClock(kCLOCK_Gpio1);
    CP_SetDutyCycle(0U);
….
}
```

To enable PWM_OUT_TRIG0, modify the above code snippet by adding the below section in the `CP_SetDutyCycle()` function:

```
    /* Setup the VAL0 trigger */
    BOARD_PWM_BASEADDR->SM[0].TCTRL |= 0x01;
```

2.4.1.4 CTIMER usage for control pilot

The CTIMER counter is used in the **EV side** of EVSE-SIG-BRD1X to measure the PWM frequency and ON time. The counter/timer is supplied with a 1 MHz clock. It captures values on the Capture registers at the rising and falling edges of the trigger input. The incoming PC_EV_DET signal is set as the capture input signal to the LPC5536/LPC55S36 MCU. Then, the edges of the PWM trigger the CTIMER to capture count values on Capture registers. The count values are then processed to calculate the frequency and ON time.

2.4.1.5 Software implementation of CTIMER

The CTIMER implementation is based on the NXP MCUXpresso SDK.

The alternative CTIMER_INP0 function is initialized as follows:

```
const uint32_t port0_pin1_config = (/* Pin is configured as CTIMER_INP0 */
          IOCON_PIO_FUNC3 |
    /* No addition pin function */
    IOCON_PIO_MODE_INACT |
    /* Standard mode, output slew rate control is enabled */
    IOCON_PIO_SLEW_STANDARD |
    /* Input function is not inverted */
    IOCON_PIO_INV_DI |
    /* Enables digital function */
    IOCON_PIO_DIGITAL_EN |
    /* Open drain is disabled */
    IOCON_PIO_OPENDRAIN_DI);
        /* PORT0 PIN1 is configured as CTIMER_INP3 */
```

CTIMER is initialized as below:

```
void CP_Init(void)
{
 /* Use 12 MHz clock for some of the Ctimers */
    CLOCK_SetClkDiv(kCLOCK_DivCtimer1Clk, 0u, false);
    CLOCK_SetClkDiv(kCLOCK_DivCtimer1Clk, 1u, true);
    CLOCK_AttachClk(kFRO_HF_to_CTIMER1);

    /* Connect CTimer capture input form CTIMER_INP0 pin */
    INPUTMUX_AttachSignal(INPUTMUX, 0U, kINPUTMUX_CtimerInp0ToTimer1Captsel);
    INPUTMUX_AttachSignal(INPUTMUX, 1U, kINPUTMUX_CtimerInp0ToTimer1Captsel);

    /* Initialize CTIMER for PWM period and ON time measurement */
    CTIMER_GetDefaultConfig(&config);

    /* Set pre-scale to run timer count @1MHz */
    config.prescale = (CP_CTIMER_CLK_FREQ/1000000) - 1;
    config.mode = kCTIMER_TimerMode;
    CTIMER_Init(CP_CTIMER, &config);

    CTIMER_RegisterCallBack(CP_CTIMER, &ctimer_callback_table[0],
 kCTIMER_MultipleCallback);
    CTIMER_SetupCapture(CP_CTIMER, kCTIMER_Capture_0, kCTIMER_Capture_RiseEdge,
 true);
    CTIMER_SetupCapture(CP_CTIMER, kCTIMER_Capture_1, kCTIMER_Capture_FallEdge,
 true);
    CTIMER_StartTimer(CP_CTIMER);
}
```

The callback functions read the Capture registers and calculate the PWM frequency and PWM ON time:

```
void ctimer_capturedOnRising_callback(uint32_t flags)
{
    countRising++;
    risingCaptureVal = CP_CTIMER->CR[0];
    TmrPeriodCounts = 0x100000000 + risingCaptureVal - lastRisingEdgeTmrVal;
    pwmFrequency = TmrPeriodCounts/1.0f;
    lastRisingEdgeTmrVal = risingCaptureVal;
}
```

```
void ctimer_capturedOnFalling_callback(uint32_t flags)
{
    countFalling++;
    fallingCaptureVal = CP_CTIMER->CR[1];
    TmrOnCounts = 0x100000000 + fallingCaptureVal - risingCaptureVal;
    pwmOnPercent = (float)TmrOnCounts/(float)TmrPeriodCounts;
    pwmOnPercentMilli = (uint16_t)(pwmOnPercent*1000); // take the integer part
    pwmOnPercent *= 100;
}
```

2.4.1.6 ADC usage for control pilot for EVSE

Irrespective of the PC_PWM state (logic high, logic low, or PWM), EVSE-SIG-BRD1X must measure the voltage level of the +12 V CP continuously in the EVSE side. First, the CP signal is buffered at the op-amp U8B for measurement. Then, the signal is divided by a resistor network of R35 (200 kΩ) and R37 (62 kΩ). This voltage has a negative half of the CP signal that is the offset from the positive bias provided by R36 (36 kΩ), which is tied to the +3.3 V supply. The ADC0 module is used in 16-bit Single-Ended mode to measure the PC_EVSE_DET voltage level.

Figure 24 explains analog-to-digital converter (ADC) triggering. The running PWM generates the hardware trigger PWM_OUT_TRIG0, which is connected internally to the INPUTMUX module. PWM_OUT_TRIG0 triggers the ADC.

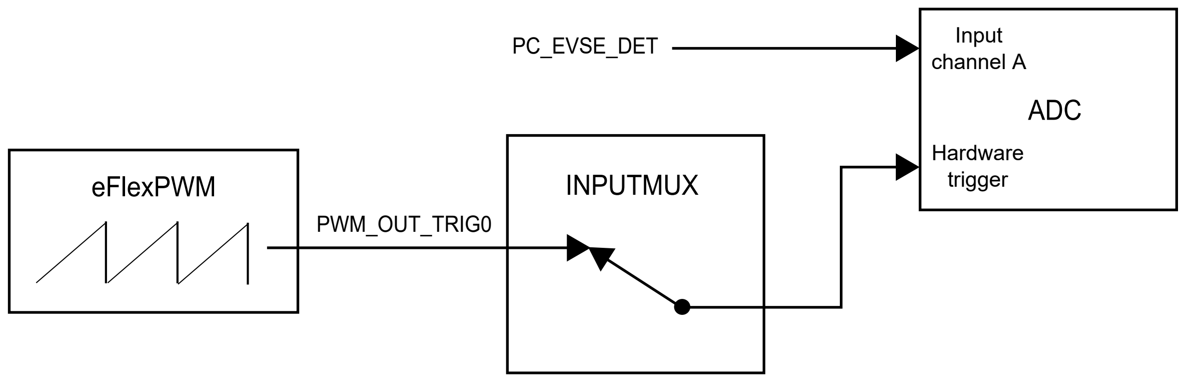

Figure 24. ADC triggering by eFlexPWM

2.4.1.7 Software implementation of ADC

The ADC implementation is based on the NXP MCUXpresso SDK.

The ADC is initialized as follows:

```
       /* setup ADC channels for PP and CP measurement */
 LPADC_GetDefaultConfig(&mLpadcConfigStruct);
 mLpadcConfigStruct.enableAnalogPreliminary = true;
#if defined(DEMO_LPADC_VREF_SOURCE)
 mLpadcConfigStruct.referenceVoltageSource = DEMO_LPADC_VREF_SOURCE;
#endif /* DEMO_LPADC_VREF_SOURCE */
#if defined(FSL_FEATURE_LPADC_HAS_CTRL_CAL_AVGS) &&
 FSL_FEATURE_LPADC_HAS_CTRL_CAL_AVGS
 mLpadcConfigStruct.conversionAverageMode = kLPADC_ConversionAverage128;
#endif /* FSL_FEATURE_LPADC_HAS_CTRL_CAL_AVGS */
 LPADC_Init(DEMO_LPADC_BASE, &mLpadcConfigStruct);

#if defined(FSL_FEATURE_LPADC_HAS_CTRL_CALOFS) &&
 FSL_FEATURE_LPADC_HAS_CTRL_CALOFS
#if defined(FSL_FEATURE_LPADC_HAS_OFSTRIM) && FSL_FEATURE_LPADC_HAS_OFSTRIM
 /* Request offset calibration. */
#if defined(DEMO_LPADC_DO_OFFSET_CALIBRATION) &&
 DEMO_LPADC_DO_OFFSET_CALIBRATION
 LPADC_DoOffsetCalibration(DEMO_LPADC_BASE);
#else
 LPADC_SetOffsetValue(DEMO_LPADC_BASE, DEMO_LPADC_OFFSET_VALUE_A,
 DEMO_LPADC_OFFSET_VALUE_B);
#endif /* DEMO_LPADC_DO_OFFSET_CALIBRATION */
#endif /* FSL_FEATURE_LPADC_HAS_OFSTRIM */
```

```
 /* Request gain calibration. */
 LPADC_DoAutoCalibration(DEMO_LPADC_BASE);
#endif /* FSL_FEATURE_LPADC_HAS_CTRL_CALOFS */

#if (defined(FSL_FEATURE_LPADC_HAS_CFG_CALOFS) &&
 FSL_FEATURE_LPADC_HAS_CFG_CALOFS)
 /* Do auto calibration. */
 LPADC_DoAutoCalibration(DEMO_LPADC_BASE);
#endif /* FSL_FEATURE_LPADC_HAS_CFG_CALOFS */

 /* Set conversion CMD configuration for CP. */
 LPADC_GetDefaultConvCommandConfig(&mLpadcCommandConfigStruct);
 mLpadcCommandConfigStruct.channelNumber = DEMO_LPADC_CP_CHANNEL;
#if defined(DEMO_LPADC_USE_HIGH_RESOLUTION) && DEMO_LPADC_USE_HIGH_RESOLUTION
 mLpadcCommandConfigStruct.conversionResolutionMode =
 kLPADC_ConversionResolutionHigh;
#endif /* DEMO_LPADC_USE_HIGH_RESOLUTION */
 mLpadcCommandConfigStruct.sampleChannelMode =
 kLPADC_SampleChannelDualSingleEndBothSide;
 LPADC_SetConvCommandConfig(DEMO_LPADC_BASE, DEMO_LPADC_CP_CMDID,
 &mLpadcCommandConfigStruct);
 /* select alt en chan 4B - Proximity Pilot PP */
 ADC0->CMD[DEMO_LPADC_CP_CMDID-1].CMDL |= ADC_CMDL_ALTBEN(DEMO_LPADC_CP_CHANNEL)
 | ADC_CMDL_ALTB_ADCH(DEMO_LPADC_PP_CHANNEL);

 /* Set trigger configuration for CP. */
 LPADC_GetDefaultConvTriggerConfig(&mLpadcTriggerConfigStruct);
 mLpadcTriggerConfigStruct.enableHardwareTrigger = true;
 mLpadcTriggerConfigStruct.targetCommandId       = DEMO_LPADC_CP_CMDID;
#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT ==
 2))
 mLpadcTriggerConfigStruct.channelAFIFOSelect = 0U;
 mLpadcTriggerConfigStruct.channelBFIFOSelect = 1U;
#endif /* FSL_FEATURE_LPADC_FIFO_COUNT */
 LPADC_SetConvTriggerConfig(DEMO_LPADC_BASE, 0U, &mLpadcTriggerConfigStruct); /*
 Configurate the trigger0. */

 /* Enable the watermark interrupt. */
#if (defined(FSL_FEATURE_LPADC_FIFO_COUNT) && (FSL_FEATURE_LPADC_FIFO_COUNT ==
 2U))
 LPADC_EnableInterrupts(DEMO_LPADC_BASE, kLPADC_FIFO0WatermarkInterruptEnable);
#else
 LPADC_EnableInterrupts(DEMO_LPADC_BASE, kLPADC_FIFOWatermarkInterruptEnable);
#endif /* FSL_FEATURE_LPADC_FIFO_COUNT */
 EnableIRQ(DEMO_LPADC_IRQn);
```

2.4.2 HomePlug Green PHY

The HomePlug Green PHY (HPGP) interface is implemented using a Lumissil CG5317 HPGP transceiver. CG5317 is HomePlug Green PHY compliant and HomePlug AV and IEEE 1901 ready. It contains an internal processor and supports the frequency band 2-30 MHz.

CG5317 has an internal analog front-end (AFE) for the medium/line interface and provides SPI slave and MII/RMII interfaces to an external host processor. Its software can be loaded through the external host processor at every power cycle or can be stored permanently to an external flash memory connected to its separate SPI master interface.

Figure 25 shows the design scheme with CG5317.

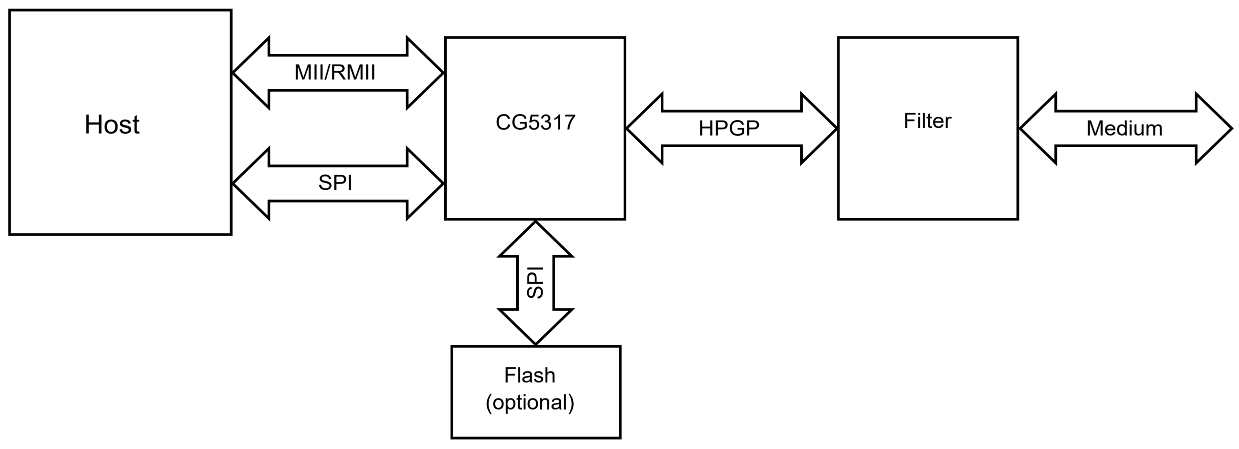

Figure 25. Design with CG5317

Note: *To get the detailed and most up-to-date information about CG5317, contact Lumissil through its website, customer support portal, or support email address (see Table 23).*

2.4.2.1 Host interface

The host interface supports two interfaces that could be used in parallel: SPI and MII/RMII.

2.4.2.1.1 SPI slave interface

Figure 26 shows the SPI host interface of CG5317.

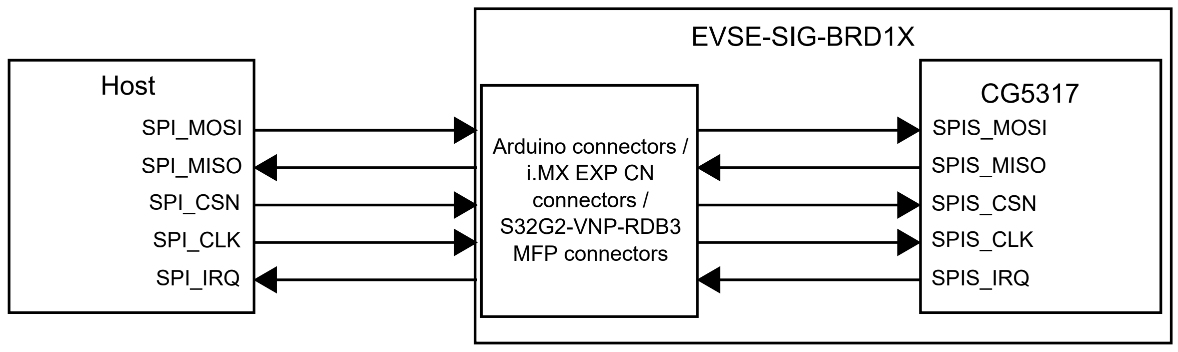

Figure 26. SPI host interface of CG5317

As shown in Figure 26, the CG5317 SPI interface provides signals for:

- Clock
- Data in
- Data out
- Chip select
- Two interrupts

Figure 27 shows the schematic design of the CG5317 SPI interface implementation.

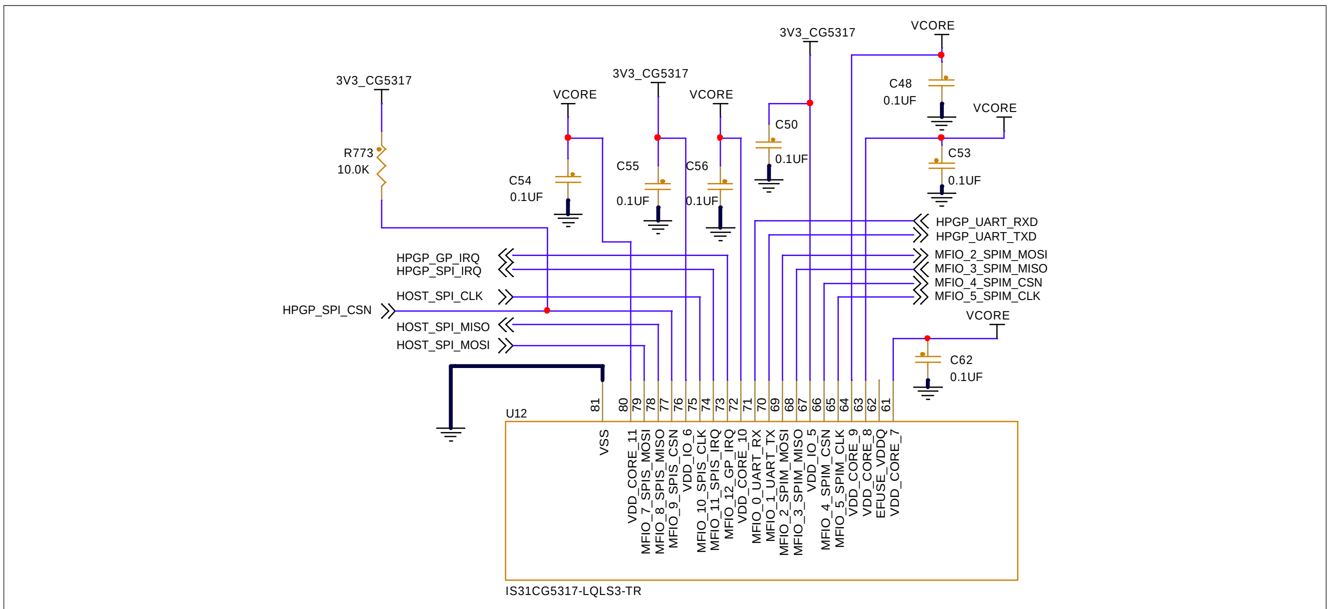

Figure 27. SPI interface design of CG5317

2.4.2.1.2 MII PHY interface

The MII PHY interface is connected to the external host as shown in Figure 28.

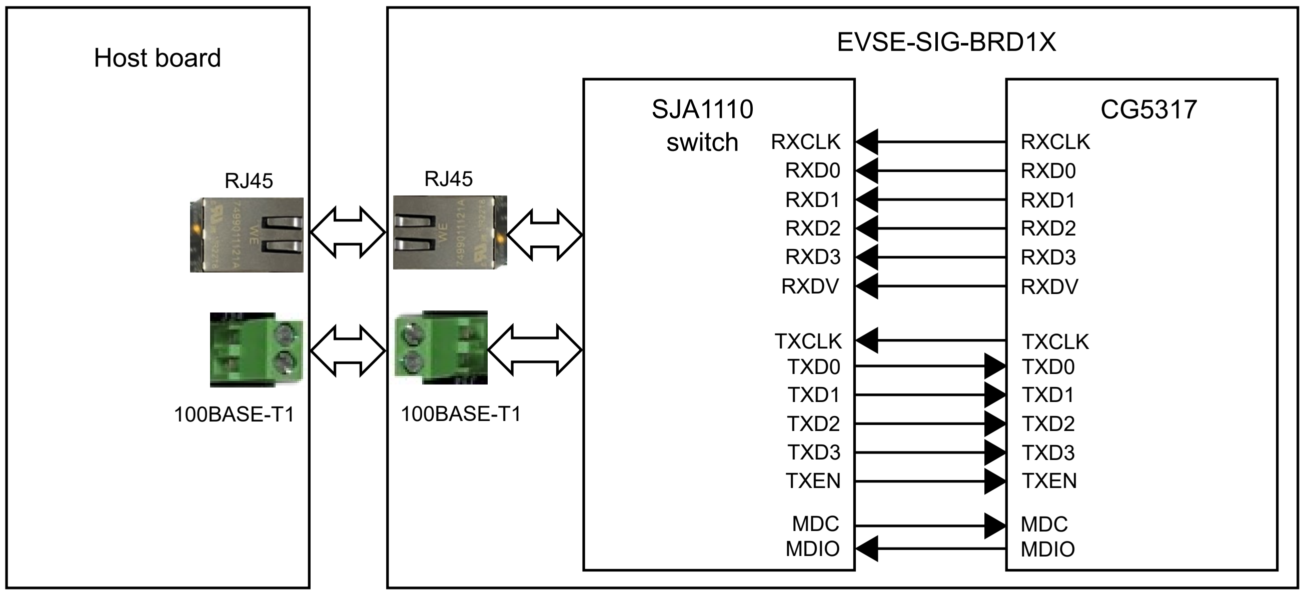

Figure 28. MII host interface of CG5317

Figure 29 shows the schematic design of the implementation in the MII interface.

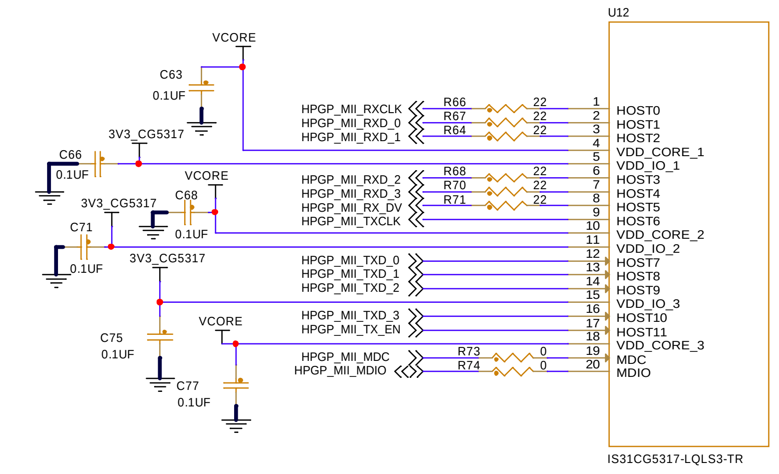

Figure 29. MII interface design of CG5317

2.4.2.2 CG5317 bootstrap

Table 10 shows the bootstrap settings of CG5317. These settings are sampled at the power cycle / reset of CG5317 and they must be set according to the expected configurations.

Table 10. CG5317 bootstrap settings

Bootstrap pin	Supported function	Control jumper	Jumper settings
STRAP0	CG5317 UART disable	J22	• Pins 1-2 shorted (strap pin value: 1): UART port cannot receive debugging messages (default setting). • Pins 2-3 shorted (strap pin value: 0): UART port can receive debugging messages.
STRAP1	CG5317 boot mode selection	J20	• Pins 1-2 shorted (strap pin value: 0): CG5317 boots from external (optional) flash. • Pins 2-3 shorted (strap pin value: 1): CG5317 boots from host (default setting).
STRAP2	CG5317 SPI bus timing mode selection (based on SPI clock polarity)	J18	• Pins 1-2 shorted (strap pin value: 1): Data is sampled on clock rising edge (SPI mode 1). • Pins 2-3 shorted (strap pin value: 0): Data is sampled on clock falling edge (SPI mode 3) (default setting).
STRAP3	CG5317 MII port address	J21 (address bit 2)	• Pins 1-2 shorted (strap pin value: 0): Address bit 2 = 0 (default setting) • Pins 2-3 shorted (strap pin value: 1): Address bit 2 = 1
STRAP4		J19 (address bit 1)	• Pins 1-2 shorted (strap pin value: 1): Address bit 1 = 1 (default setting) • Pins 2-3 shorted (strap pin value: 0): Address bit 1 = 0
STRAP5		J17 (address bit 0)	• Pins 1-2 shorted (strap pin value: 1): Address bit 0 = 1 (default setting) • Pins 2-3 shorted (strap pin value: 0): Address bit 0 = 0

2.4.2.3 CG5317 UART debug port

The UART port of CG5317 is used as a debug port for the host interface. To route the UART TXD and RXD signals between CG5317 and the host connectors, pins 2 and 3 must be shorted for each of the onboard jumpers J41 and J42. To enable UART debug, the bootstrap STRAP0 (UART_DISABLE) must be set to "enable" position. After the CG5317 is brought out of reset by deasserting its RESET signal, messages can be observed through the UART port of the host connectors.

2.4.2.4 CG5317 analog interface

To get the detailed and most up-to-date information about CG5317, contact Lumissil through its website, customer support portal, or support email address (see Table 23).

2.5 GFCI circuit

The ground-fault circuit interrupter (GFCI) is a fast operating charging circuit breaker. As the charging stations can be installed in open areas or outdoor environments, leakage to the watery surface of the earth may cause electric shock. In such cases, there is a difference between the phase current and the neutral current through the conductors of the AC supply. Therefore, the charging station must be equipped with the GFCI circuit. It is designed to trigger the generation circuit, which is sent to the relay drive circuit and also to the PIO/interrupt of the LPC5536/LPC55S36 MCU. The trigger to the relay driver circuit enables a real-time response from the EVSE to disconnect the AC supply for user safety.

Figure 30 shows the block schematic of an external GFCI sensor coil that is connected to the GFCI detection circuit on the board. The figure also shows how the GFCI sensor coil interacts with the relay driver circuit and the LPC5536/LPC55S36 MCU.

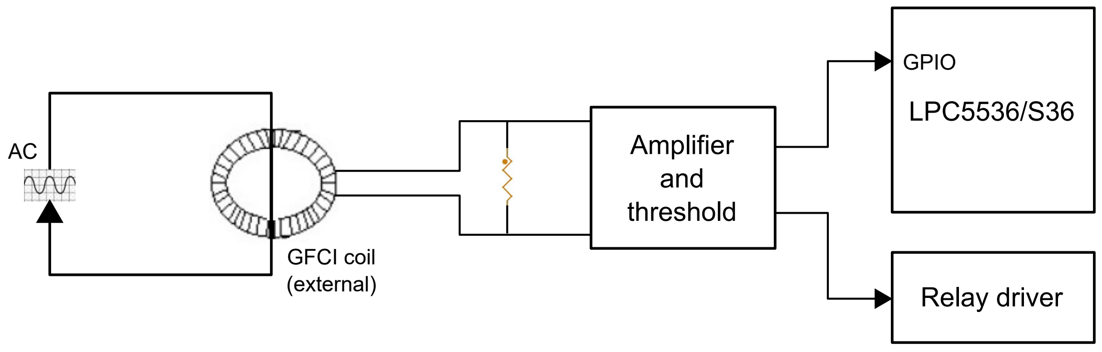

Figure 30. GFCI detection circuit

2.5.1 Schematic design

Figure 31 shows the design implementation of the GFCI circuit.

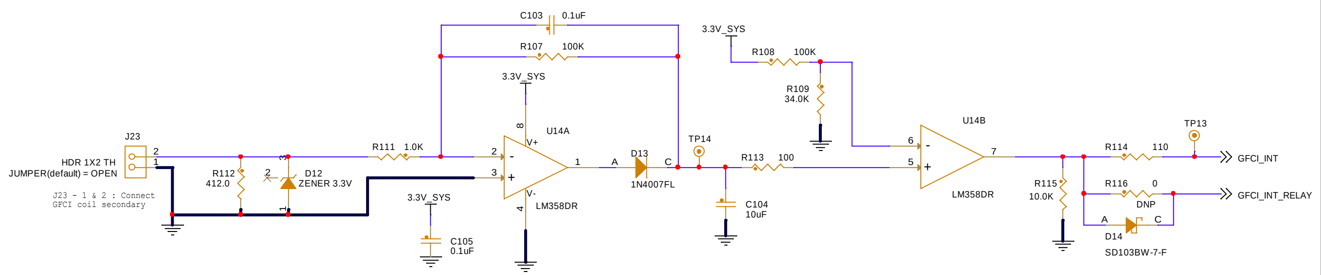

Figure 31. Design of GFCI detection circuit

An external GFCI coil or current transformer (CT) is used as a sensor for ground fault. All the phase and neutral conductors of the AC are passed through the GFCI coil. The AC conductors act as the primary side of the transformer while the coil output acts as the secondary side.

In the absence of a shock or leakage condition, the currents passing through the phase and neutral conductors cancel the induction of each other. Therefore, no induced current is present at any end of the GFCI coil. However, if there is a shock condition, a small amount of current flows to the surface of the earth. Therefore, less current flows through the neutral return path conductor. It results in little induced current between the ends of the GFCI coil.

Then, both ends of the coil are connected to the J23 connector on the board that is connected to a burden resistor R112. The secondary current induced in the coil produces little potential difference across the burden resistor. It is then fed to an inverting amplifier made from the op-amp U14A.

The op-amp saturates quickly enough to react to the induced input AC signal. A discharge path is also created so that the op-amp output is not stuck high even when the input fault condition disappeared. To interface the fault level to the MCU, the output of U14A is fed to an op-amp comparator U14B. Then, the output is ready to trigger an interrupt, preferably through the GPIO pin of the MCU.

Also, the output of the GFCI fault detection circuit is fed to the relay driver circuit, as shown in Figure 32. It ensures quick disconnect of the AC supply, preventing any shock or damage.

2.5.2 Software implementation

The GFCI software implementation on the EVSE side is based on the NXP MCUXpresso SDK.

GPIO and the interrupt modules can be initialized for GFCI as shown below:

```
void GFCI_Init(void)
{
    /* Initialize PINT */
    PINT_Init(PINT);
    /* Setup Pin Interrupt 0 for rising edge */
    PINT_PinInterruptConfig(PINT, kPINT_PinInt0, kPINT_PinIntEnableBothEdges,
 pint_intr_callback);
    /* Enable callbacks for PINT0 by Index */
    PINT_EnableCallbackByIndex(PINT, kPINT_PinInt0);
}
```

The GFCI events are registered in the interrupt handler:

```
void pint_intr_callback(pint_pin_int_t pintr, uint32_t pmatch_status)
{
    /* GFCI interrupt */
    g_GFCIOccurred = true;
}
```

The main program can check the GFCI occurrence status by invoking the following function:

```
void GFCI_Process(void)
{
    if (g_GFCIOccurred)
    {
        gfciValue = GPIO_PinRead(GPIO, BOARD_GFCI_INT_PORT, BOARD_GFCI_INT_PIN);
        g_GFCIOccurred = false;
        PRINTF("A GFCI occurred/restored event.\n");
    }
}
```

Note: *Immediately after the GFCI fault is detected, the relay driver input circuit triggers the relay driver. The host can read the event from the* `GFCI_Process()` *function. Initially, LED1 (D18) remains OFF. However, when a GFCI fault is detected, it starts blinking at twice the rate of LED2 (D19).*

2.6 Relay driver circuit

2.6.1 Block diagram

In EVSE-SIG-BRD1X, the relay driver circuit can drive two DC coil relays. The relays can turn ON or turn OFF an AC supply of single-phase to three-phase connected to the EV through the charging cable. The external relays are hosted in the EVSE system. EVSE-SIG-BRD1X drives an external relay using a host controller command or when the GFCI circuit triggers it. An additional emergency stop push button is also supported. Figure 32 shows the block schematic of the design.

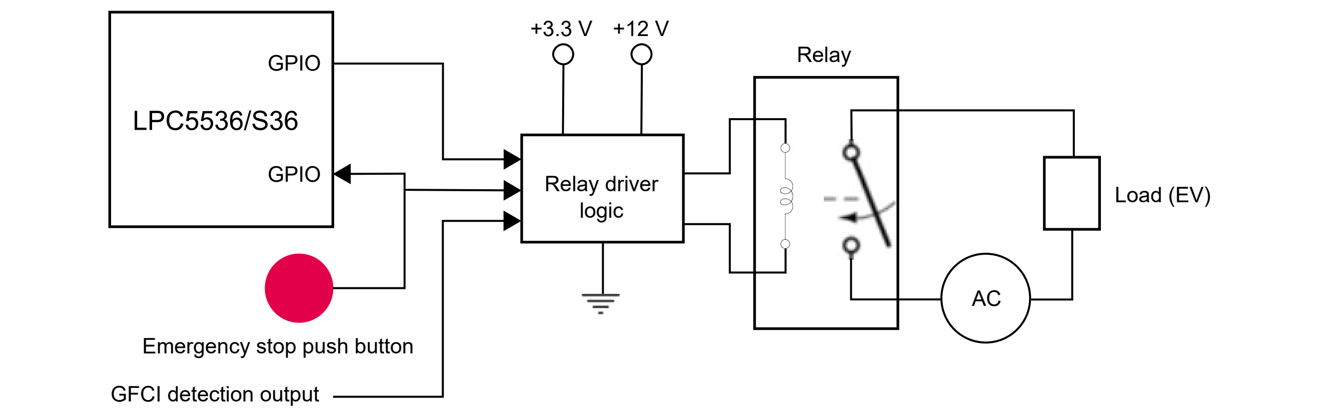

Figure 32. Relay driver circuit

2.6.2 Schematic design

Figure 33 shows the hardware implementation of the relay driver circuit.

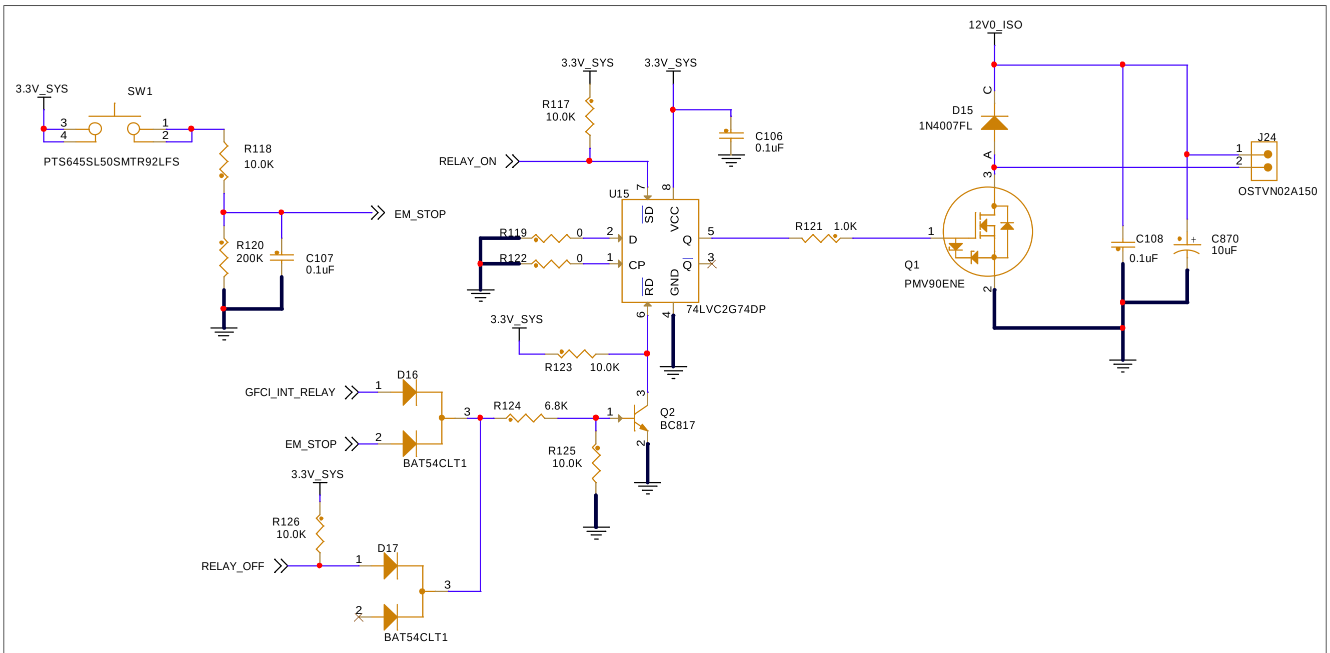

Figure 33. Relay driver circuit design

A D-type flip-flop (U15) is used for the logic implementation of the relay driver circuit. At power-on reset, the MCU GPIO pins RELAY_ON and RELAY_OFF are at tri-state and the other circuit signals EM_STOP and GFCI_INT_RELAY are at low state. The pull-up register R125 turns ON the NPN transistor Q2, which in turn pulls down the reset signal of the U15 flip-flop. It ensures that the default state of the flip-flop output Q is low at power-up.

To turn ON the relay of the charging system, set the output of the U15 flip-flop as follows:

1. Drive the MCU GPIO pin RELAY_OFF to low state.
2. Drive the MCU GPIO pin RELAY_ON to high state.

It in turn withdraws the Reset pin and sets the Set pin of the D flip-flop, resulting in setting the flip-flop output to the high level. The output of the flip-flop is fed to the gate input of the N-channel MOSFET Q1. An external relay coil is connected at J24 that acts as a load impedance between its drain terminal and the 12 V supply voltage. The external relay should be a DC coil that typically draws a current of 140 mA. The MOSFET and onboard 12 V power supply can drive up to two such external relays connected across J24. It is sufficient for an AC charging application with up to three phases.

To turn OFF the relay, clear the flip-flop output as follows:

1. Drive RELAY_ON to low state.
2. Drive RELAY_OFF to high state.

The flip-flop output can also be cleared to turn OFF the relay produced from the GFCI coil output. GFCI_INT_RELAY is usually at the low voltage. However, if it is at the high level, it can reset the flip-flop logic to the low output state, turning OFF the relay.

The EM_STOP button simulates a manual push button. It can be used to turn OFF the relay during an emergency.

2.6.3 Software implementation

The relay driver software implementation on the EVSE side is based on the NXP MCUXpresso SDK.

The initialization function initializes the data and state variables only:

```
void Relay_Init(void)
{
    emStopValue = 0;
    relayClosedState = false;
    oldRelayClosedState = false;
    relayOpenedState = false;
}
```

Relay can be opened (OFF) using the `Relay_Open()` function:

```
void Relay_Open(void)
{
    GPIO_PortSet(GPIO, BOARD_RELAY_OFF_PORT, 1u << BOARD_RELAY_OFF_PIN); //
 assert RELAY_OFF
    SDK_DelayAtLeastUs((3000), BOARD_BOOTCLOCKPLL150M_CORE_CLOCK);
    GPIO_PortClear(GPIO, BOARD_RELAY_OFF_PORT, 1u << BOARD_RELAY_OFF_PIN); //
 withdraw RELAY_OFF
    SDK_DelayAtLeastUs((3000), BOARD_BOOTCLOCKPLL150M_CORE_CLOCK);
    relayClosedState = false;
    relayOpenedState = true;
}
```

Relay can be closed (ON) using the `Relay_Close()` function:

```
void Relay_Close(void)
{
    GPIO_PortClear(GPIO, BOARD_RELAY_ON_PORT, 1u << BOARD_RELAY_ON_PIN); //
 assert RELAY_ON
    SDK_DelayAtLeastUs((3000), BOARD_BOOTCLOCKPLL150M_CORE_CLOCK);
    GPIO_PortSet(GPIO, BOARD_RELAY_ON_PORT, 1u << BOARD_RELAY_ON_PIN); //
 withdraw RELAY_ON
    relayClosedState = true;
    relayOpenedState = false;
}
```

2.7 LPC5536/LPC55S36 MCU

EVSE-SIG-BRD1X hosts an LPC5536/LPC55S36 controller to support the required local controller functions of the board. This MCU acts as a utility controller for the EVSE system. It performs the following functions during the EVSE or EV simulation of EVSE-SIG-BRD1X:

- Generates control pilot PWM using eFlexPWM.
- Measures voltage level in the EVSE Simulation mode of EVSE-SIG-BRD1X, using the ADC module.
- Measures the frequency and duty cycle of the control pilot signal in the EV simulation of EVSE-SIG-BRD1X, using the CTIMER module.
- Measures proximity pilot level using the ADC module.
- Detects GFCI fault driven by an interrupt or at GPIO level.
- Controls relay ON/OFF GPIO function.
- Provides UART communication port between host controller (through host connector interfaces) and the EVSE-SIG-BRD1X MCU. In such a case, it acts as a slave processor for the master processor on the host controller board. For example, in the EVSE Simulation mode, the LPC5536/LPC55S36 MCU can set the control pilot state to high, low, or PWM, based on the host controller request received through the UART interface. For EVSE simulation of the board, UART is the default channel of communication.

- Provides a LIN slave communication port between host controller (through host connector interfaces) and the EVSE-SIG-BRD1X MCU. For EV simulation of the board, LIN is the default channel of communication.
- Supports CAN interface as a future expansion option for CAN communication.

2.7.1 Block diagram

Figure 34 shows the design using LPC5536/LPC55S36.

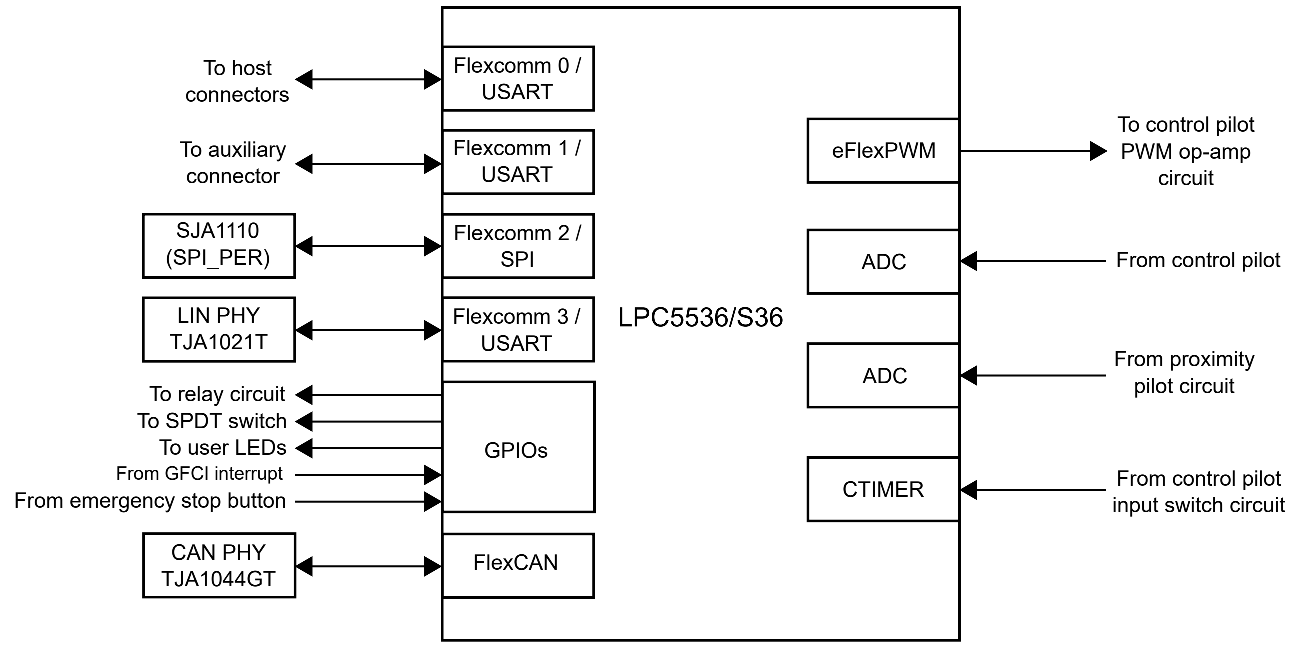

Figure 34. LPC5536/LPC55S36 interfaces on board

2.7.2 Schematic design

Figure 35 shows the schematic design with LPC5536/LPC55S36.

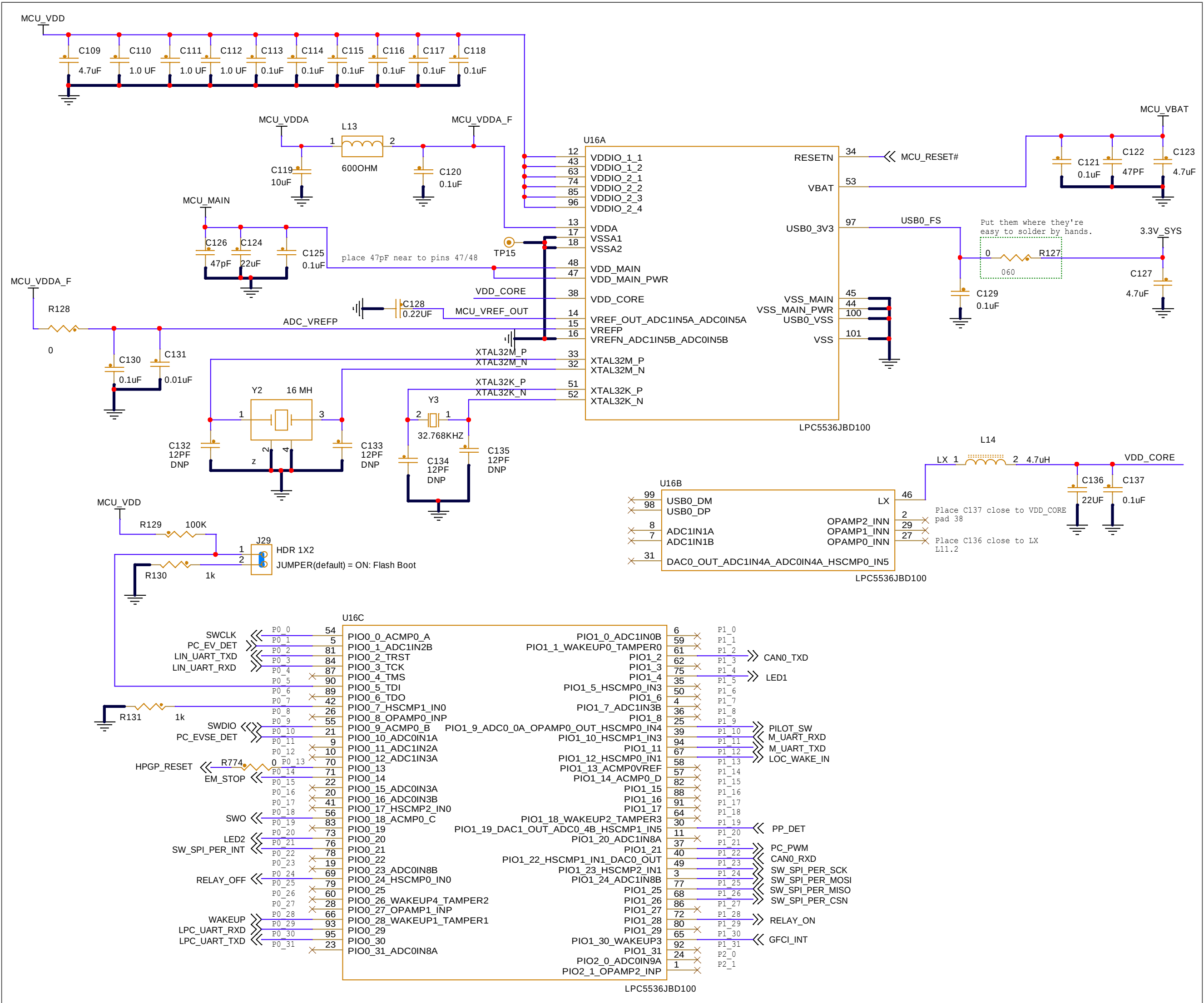

Figure 35. LPC5536/LPC55S36 schematic design

2.7.3 LPC5536/LPC55S36 pin usage

Table 11 shows the list of non-power MCU pin functions used in EVSE-SIG-BRD1X.

Table 11. LPC5536/LPC55S36 pin usage

Pin name	Signal name	MCU block usage	Function
PIO0_0_ACMO0_A	SWCLK	Serial wire debug	Serial wire debug clock
PIO0_1_ADC1IN2B	PC_EV_DET	CTIMER CAPTURE IN	Control pilot detection in the EV mode
PIO0_2_TRST	LIN_UART_TXD	UART transmit signal to the LIN transceiver	
PIO0_3_TCK	LIN_UART_RXD	UART receive signal from the LIN transceiver	
PIO0_5_TMS		ISP	ISP boot selection

Table 11. LPC5536/LPC55S36 pin usage*...continued*

Pin name	Signal name	MCU block usage	Function
PIO0_7_HSCMP1_IN0		ISP	ISP boot selection
PIO0_9_ACMP0_B	SWDIO	Serial wire debug	Serial wire debug input/output
PIO0_10_ADC0IN1A	PC_EVSE_DET	ADC	Pilot control sense
PIO0_13	HPGP_RESET	GPIO	CG5317 reset signal
PIO0_14	EM_STOP	GPIO	Emergency stop button press detection
PIO0_18_ACMP0_C	SWO	Serial wire debug	Serial wire debug output
PIO0_20	LED2	GPIO	User LED
PIO0_21	SW_SPI_PER_INT	GPIO	SJA1110B SPI PER interface interrupt
PIO0_24_HSCMP0_IN0	RELAY_OFF	GPIO	Relay OFF control
PIO0_28_WAKEUP1_TAMPER1	WAKEUP	GPIO	Wake-up signal from proximity pilot detection
PIO0_29	LPC_UART_RXD	Flexcomm/USART	UART receive
PIO0_30	LPC_UART_TXD	Flexcomm/USART	UART transmit
PIO1_2	CAN0_TXD	FlexCAN	CAN transmit
PIO1_4	LED1	GPIO	User LED
PIO1_9_ADC0_0A	PILOT_SW	GPIO	Control pilot single pole double through (SPDT) switch
PIO1_10_HSCMP_IN3	M_UART_RXD	Flexcomm/USART	UART receive
PIO1_11	M_UART_TXD	Flexcomm/USART	UART transmit
PIO1_12	LOC_WAKE_IN	GPIO	Local wake up to SJA1110B switch
PIO1_19_DAC1_OUT_ADC0_4B	PP_DET	ADC	Proximity pilot detection input
PIO1_21	PC_PWM	eFlexPWM	Pilot control PWM output
PIO1_22	CAN0_RXD	FlexCAN	CAN receive
PIO1_23	SW_SPI_PER_SCK	Flexcomm/SPI	SPI master to SJA1110B slave clock
PIO1_24	SW_SPI_PER_MOSI	Flexcomm/SPI	SPI master to SJA1110B slave data out
PIO1_25	SW_SPI_PER_MISO	Flexcomm/SPI	SPI master to SJA1110B slave data in
PIO1_26	SW_SPI_PER_CSN	Flexcomm/SPI	SPI master to SJA1110B slave chip select
PIO1_28	RELAY_ON	GPIO	Relay ON control
PIO1_30_WAKEUP3	GFCI_INT	GPIO	GFCI event interrupt
XTAL_32M_[P, M]	XTAL_32M_[P, M]	Clock	High-frequency external crystal oscillator pins
XTAL_32K_[P, M]	XTAL_32K_[P, M]	Clock	Low-frequency external crystal oscillator pins

2.7.4 LPC5536/LPC55S36 boot options

EVSE-SIG-BRD1X uses In-System Programming (ISP) through the UART interface to program LPC5536/ LPC55S36. UART peripheral implements auto-baud detection. To set up LPC5536/LPC55S36 for ISP programming, the ISP mode selection jumper J29 setting must be changed from shorted (default setting) to open.

Table 12 shows the settings of jumper J29 for boot mode selection.

Table 12. LPC5536/LPC55S36 boot mode selection in EVSE-SIG-BRD1X

Jumper 29 setting	Boot mode
Open	ISP boot
Shorted (default setting)	Internal flash boot

2.7.5 Debug interface

EVSE-SIG-BRD1X provides a serial wire debug (SWD) port through connector J30 for debugging the LPC5536/ LPC55S36 MCU. You can use a MCU-Link debug probe, PEmicro debug probe, or MCU-Link Pro debug probe to program and debug the MCU.

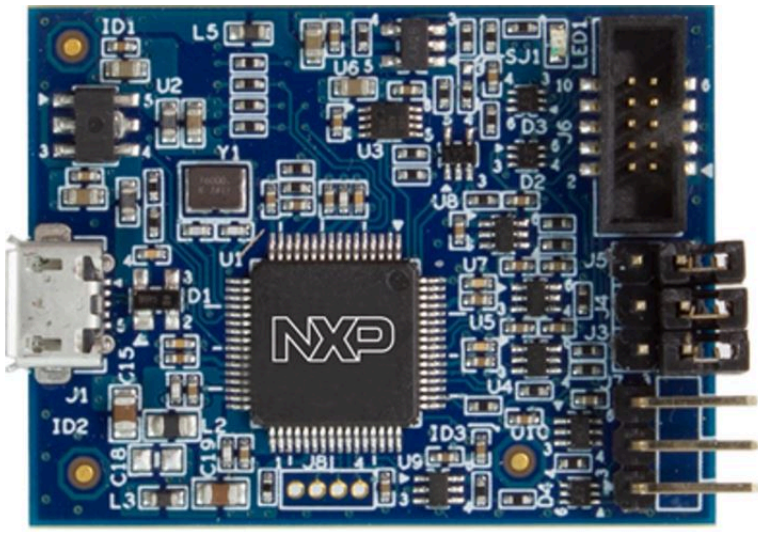

Figure 36. MCU-Link debug probe

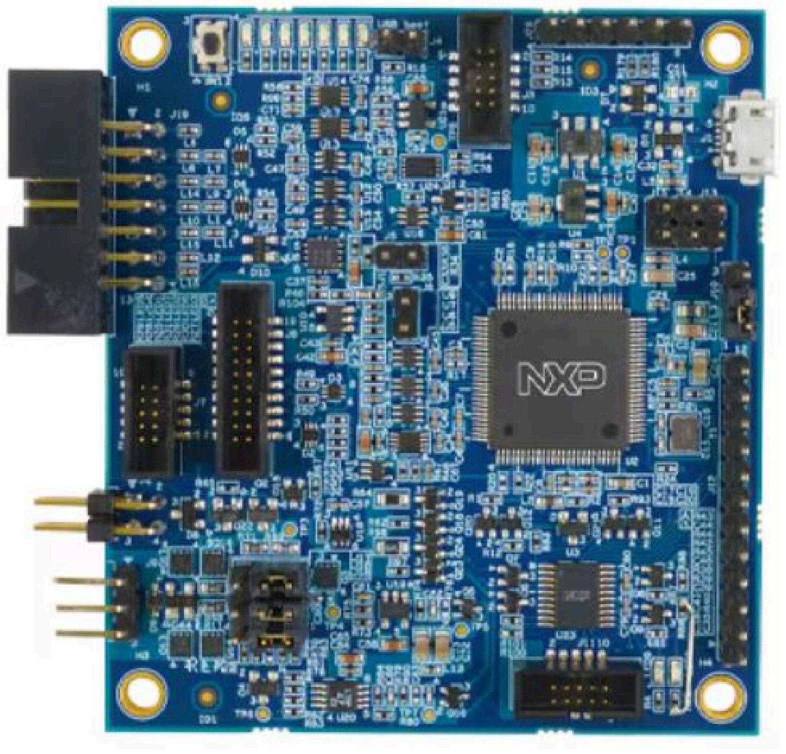

Figure 37. MCU-Link Pro debug probe

2.8 SJA1110B switch

EVSE-SIG-BRD1X hosts an NXP SJA1110B automotive Ethernet switch, which allows a host controller to connect to the HPGP through an Ethernet interface. SJA1110B provides the following two Ethernet interfaces for host Ethernet connections:

- 100BASE-T1
- 100BASE-TX

To program the SJA1110B switch, connect a suitable external programmer to the onboard SWD connector. During factory programming, the SJA1110B switch is pre-programmed to a default working state, and it rarely needs reprogramming.

2.8.1 Block diagram

Figure 38 shows the design using the SJA1110B switch.

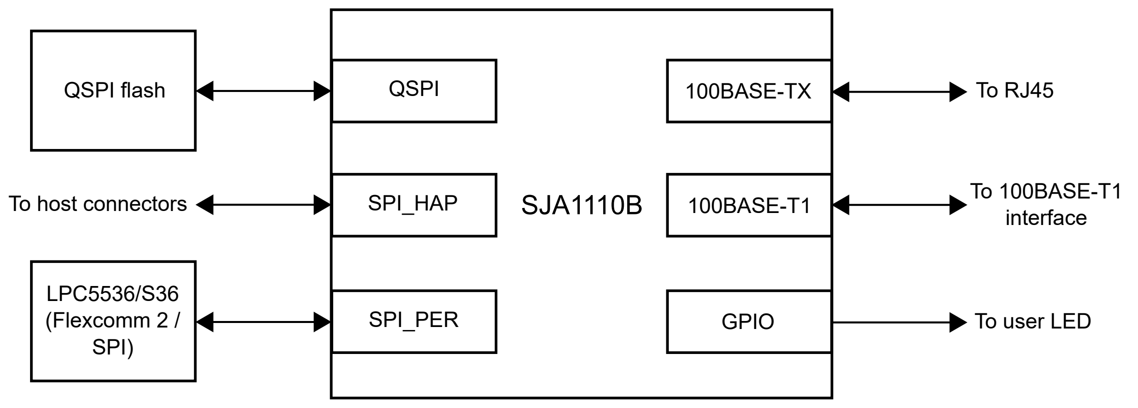

Figure 38. SJA1110B interfaces on board

The SJA1110B switch is an automotive Ethernet switch that integrates:

- Five IEEE 100BASE-T1 PHYs
- A single IEEE 100BASE-TX PHY
- Two MII/RMII/RGMII interfaces
- Two SGMII interfaces
- An Arm Cortex-M7-core-based host controller
- A SPI_HOST interface
- A SPI_PER interface

The SJA1110B switch is used in the design as follows:

- One port of 100BASE-T1 provides an Ethernet host interface to the CG5317 HPGP.
- One port of 100BASE-TX provides an Ethernet host interface to the CG5317 HPGP.
- One port of the MII interface is available for the CG5317 HPGP.
- An Arm Cortex-M7 core based host controller is used to run the switch firmware.
- The SPI_HOST interface allows firmware download through an external SPI master.
- The SPI_PER interface allows communication with the onboard LPC5536/LPC55S36 MCU.

2.8.2 Schematic design

Figure 39 shows the schematic design using SJA1110B.

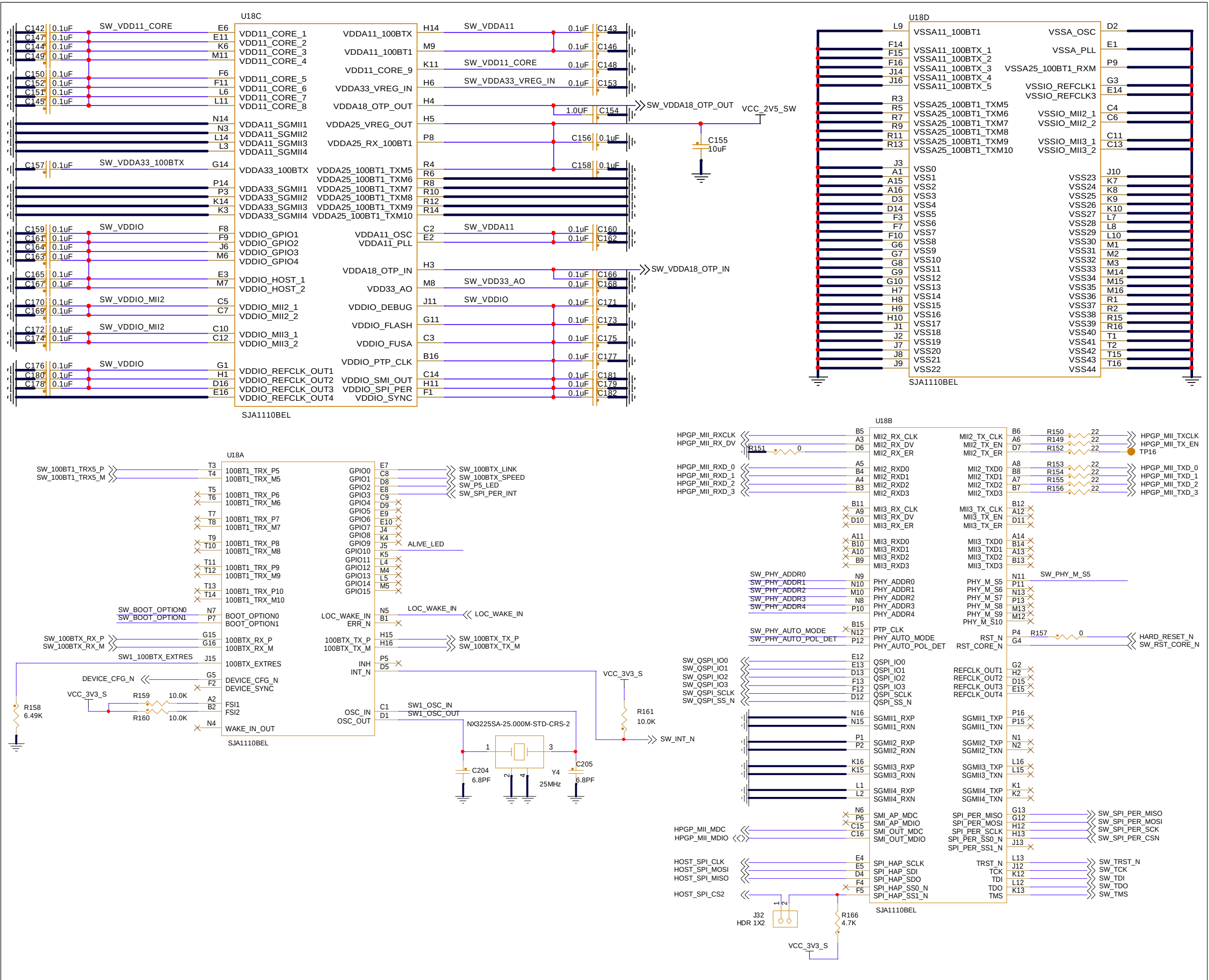

Figure 39. SJA1110B hardware design on board

2.8.3 SJA1110B pin usage

Table 13 shows the usage of non-power pins of SJA1110B in the design.

Table 13. SJA1110B pin usage

Pin name	Signal name	MCU block usage	Function
100BT1_TRX_P5	SW_100BT1_TRX5_P	100BASE-T1 port 5 terminal	100BASE-T1 Ethernet port
100BT1_TRX_M5	SW_100BT1_TRX5_M		
BOOT_OPTION[1:0]	SW_BOOT_OPTION[1:0]	Boot	Boot option selection
100BTX_RX_P	SW_100BTX_RX_P	100BASE-TX terminal	100BASE-TX Ethernet port
100BTX_RX_M	SW_100BTX_RX_M		
100BTX_TX_P	SW_100BTX_TX_P		
100BTX_TX_M	SW_100BTX_TX_M		

All information provided in this document is subject to legal disclaimers.

© 2024 NXP B.V. All rights reserved.

Table 13. SJA1110B pin usage*...continued*

Pin name	Signal name	MCU block usage	Function
DEVICE_CFG_N	DEVICE_CFG_N	Switch subsystem	Drive switch subsystem configuration completion LED
GPIO0	SW_100BTX_LINK	GPIO	Drive link LED for 100BASE-TX
GPIO1	SW_100BTX_SPEED	GPIO	Drive speed indication LED for 100 BASE-TX
GPIO2	SW_P5_LED	GPIO	Drive link status LED for 100BASE-T1 port 5
GPIO3	SW_SPI_PER_INT	GPIO	Interrupt connected to the LPC5536/LPC55S36 MCU for SPI packet error rate (PER) testing
GPIO10	ALIVE_LED	GPIO	Drive LED to indicate that SJA1110 B is up and running
LOC_WAKE_IN	LOC_WAKE_IN	Power supply management	Wakes up SJA1110B when asserted
INT_N	SW_INT_N	Host access point	Interrupt output
OSC_[IN,OUT]	SW1_OSC_[IN,OUT]	Clock	High-frequency crystal oscillator pins
MII2_RXCLK	HPGP_MII_RXCLK	MII	MII receive clock
MII2_RX_DV	HPGP_MII_RX_DV	MII	MII received data valid
MII2_RXD[3:0]	HPGP_MII_RXD_[3:0]	MII	MII received data
MII2_TX_CLK	HPGP_MII_TXCLK	MII	MII transmit clock
MII2_TX_EN	HPGP_MII_TX_EVN	MII	MII transmit data enable
MII2_TXD[3:0]	HPGP_MII_TXD_[3:0]	MII	MII transmit data
PHY_ADDR[4:0]	SW_PHY_ADDR[4:0]	100BASE-T1	Set the base PHY address for all 100BASE-T1 PHYs
PHY_AUTO_MODE	SW_PHY_AUTO_MODE	100BASE-T1	Automatic polarity detection
PHY_AUTO_POL_ DET	SW_PHY_AUTO_MODE_DET	100BASE-T1	Automatic mode select
QSPI_IO[3:0]	SW_QSPI_IO[3:0]	QSPI	QSPI data
QSPI_SCLK	SW_QSPI_SCLK	QSPI	QSPI clock
QSPI_SS_N	SW_QSPI_SS_N	QSPI	QSPI chip select
SMI_OUT_MDC	HPGP_MII_MDC	SMI	SMI clock
SMI_OUT_MDIO	HPGP_MII_MDI	SMI	SMI data
SPI_HAP_SCLK	HPST_SPI_CLK	SPI_HAP	SPI_HAP clock
SPI_HAP_SDI	HOST_SPI_MOSI	SPI_HAP	SPI_HAP data input
SPI_HAP_SDO	HOST_SPI_MISO	SPI_HAP	SPI_HAP data output
SPI_HAP_SS1_N	HOST_SPI_CS2	SPI_HAP	SPI_HAP chip select
PHY_M_S5	SW_PHY_SW_M5	100BASE-T1 setting	100BASE-T1 master/slave setting
RST_N	HARD_RESET_N	Switch	Reset input for entire switch

Table 13. SJA1110B pin usage*...continued*

Pin name	Signal name	MCU block usage	Function
RST_CORE_N	SW_RST_CORE_N	Switch core	Reset input for the digital core of the switch
SPI_PER_MISO	SW_SPI_PER_MISO	SPI_PER	SPI interface MOSI signal to LPC5536/LPC55S36
SPI_PER_MOSI	SW_SPI_PER_MOSI	SPI_PER	SPI interface MISO signal to LPC5536/LPC55S36
SPI_PER_SCLK	SW_SPI_PER_SCK	SPI_PER	SPI interface clock signal to LPC5536/LPC55S36
SPI_PER_SS0_N	SW_SPI_PER_CSN	SPI_PER	SPI interface chip select to LPC5536/LPC55S36
TRST_N	SW_TRST_N	JTAG / serial wire debug	JTAG reset
TCK	SW_TCK	JTAG / serial wire debug	Serial wire debug clock
TDI	SW_TDI	JTAG / serial wire debug	JTAG TDI
TDO	SW_TDO	JTAG / serial wire debug	JTAG TDO / serial wire debug SWO
TMS	SW_TMS	JTAG / serial wire debug	JTAG TMS / serial wire debug SWDIO

2.8.4 SJA1110B bootstrap

EVSE-SIG-BRD1X provides a DIP switch SW2 for controlling the power-on bootstrap functions of the SJA1110B switch. For more details, see Table 5.

2.8.5 SJA1110B software

The SJA1110B software has been developed using S32 Design Studio (S32DS) IDE and S32 SDK for SJA1110 RTM 1.0.2. The details of the software implementation are out of the scope of this document. For more information, contact NXP technical support / community support on https://www.nxp.com/.

2.8.6 Debug interface

EVSE-SIG-BRD1X provides a debug port through connector J33 for debugging the SJA1110B Ethernet switch. You can use a PEmicro or Lauterbach debugger to program and debug the SJA1110B switch.

2.9 UART interface

EVSE-SIG-BRD1X provides two universal asynchronous receiver/transmitter (UART) ports: host UART and auxiliary UART. These ports are connected to the LPC5536/LPC55S36 MCU. The two UART ports are shown in Figure 40.

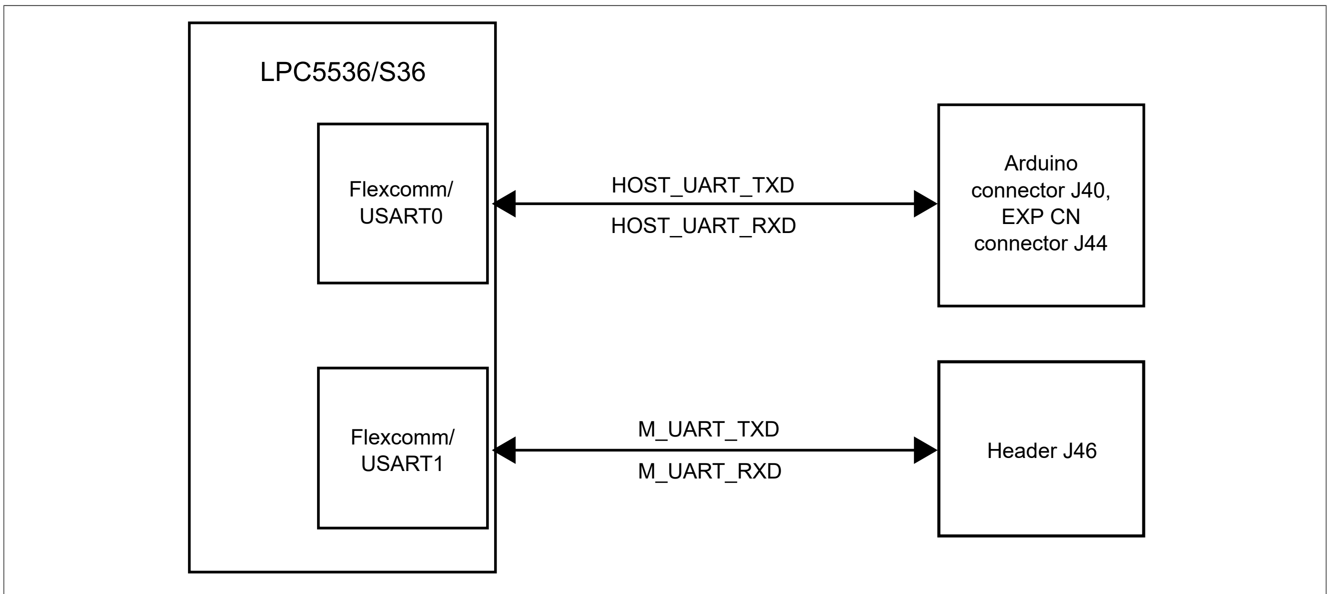

Figure 40. UART ports

The host UART port is available over Arduino and EXP CN host connectors, as described in Table 14.

Table 14. Host UART port

Host UART pin	Locations in board
HOST_UART_TXD	• J40 pin 1 • J44 pin 10
HOST_UART_RXD	• J40 pin 2 • J44 pin 8

The auxiliary UART port is provided to meet additional UART requirements, for example, for sending a debug out message from the board to another UART target. Table 15 describes the auxiliary UART port.

Table 15. Auxiliary UART port

Auxiliary UART pin	Location in board
M_UART_TXD	J46 pin 2
M_UART_RXD	J46 pin 3

2.9.1 Software implementation

The two UART ports are initialized for serial communication. Flexcomm USART0 port is available for use with the following configuration:

- Baud rate: 115200
- Data: 8 bits
- Parity: None
- Stop: 1 bit

```
/*!
* @brief Initializes the UART instances which are required for communication
* with host controller board.
*/
void Uart_ModuleInit(void)
```

```
{
    usart_config_t usartConfig;
    /* Host UART */
    /*
    * uartConfig.baudRate_Bps = 115200;
    * uartConfig.parityMode = kUART_ParityDisabled;
    * uartConfig.stopBitCount = kUART_OneStopBit;
    * uartConfig.txFifoWatermark = 0;
    * uartConfig.rxFifoWatermark = 1;
    * uartConfig.enableTx = false;
    * uartConfig.enableRx = false;
    */
    USART_GetDefaultConfig(&usartConfig);
    usartConfig.baudRate_Bps = 115200;
    usartConfig.enableTx = false;
    usartConfig.enableRx = true;
    USART_Init(HOST_USART_BASE_PTR_CONTROL, &usartConfig,
 HOST_USART_CONTROL_CLOCK);
    /* Enable RX interrupt. */
    USART_EnableInterrupts(HOST_USART_BASE_PTR_CONTROL,
 kUSART_RxLevelInterruptEnable);
    uartRxPortStatus[UART_CONTROL_INDEX] = uartTxPortStatus[UART_CONTROL_INDEX]
 = UART_IDLE;
    uartRxBufIndex[UART_CONTROL_INDEX] = 0;
    uartTxBufIndex[UART_CONTROL_INDEX] = 0;
    NVIC_SetPriority(FLEXCOMM0_IRQn, HOST_USART_INTERRUPT_PRIORITY);
    EnableIRQ(FLEXCOMM0_IRQn);
}
```

The `Comm_Process()` function processes the commands received in a ring buffer and sends an appropriate response code for each command.

2.9.2 UART commands

The host UART serial interface can receive commands from the host controller to read and write parameters on EVSE-SIG-BRD1X. There is a pre-defined set of commands supported at the UART interface. After receiving commands from the UART interface, the LPC5536/LPC55S36 MCU on the board responds to the supported commands, as explained in Figure 41. The LPC5536/LPC55S36 MCU sends UNACK for the unsupported commands.

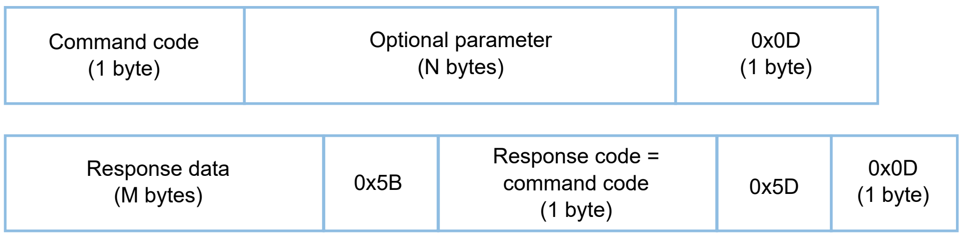

Figure 41. UART command response format

As shown in Figure 41, each command has a command code, an optional parameter, and a command end delimiter.

The host controller must wait until it gets the response of the last sent command, as the UART interface implementation of the EVSE-SIG-BRD1X software does not support a command queue. However, the user can change the implementation to add support for a command queue.

Table 16 lists supported commands and their responses. Some of the commands are applicable only to the EVSE side or the EV side of the software.

Table 16. Supported UART commands

Type/side	Command code (sent by host)	Command parameter	Response data (sent by EVSE-SIG-BRD1X)	Response code	Description	Example
Read / EVSE, EV	0x62 (alphabet 'b')	No parameter, 0 bytes	PP state, 1 byte	'b'	Read proximity pilot state	• Command: "b\r" • Response: – "0[b]\r": The coupler is not connected to the inlet. Therefore, PP is not detected. – "1[b]\r": The coupler is connected but the latch switch is open. – "2[b]\r": The coupler is connected and the latch switch is closed.
Read / EVSE	0x63 (alphabet 'c')	No parameter, 0 bytes	CP state, 1 byte	'c'	Read control pilot state	• Command: "c\r" • Response: – "0[c]\r": Control pilot line voltage is at +12 V. – "1[c]\r": Control pilot line voltage is at +9 V. – "2[c]\r": Control pilot line voltage is at +6 V. – "3[c]\r": Control pilot line voltage is at +3 V. – "4[c]\r": Control pilot line voltage error – "5[c]\r": Control pilot line voltage indicates that EVSE is offline.
Read / EVSE	0x64 (alphabet 'd')	No parameter, 0 bytes	GFCI state, 1 byte	'd'	Read GFCI state	• Command: "d\r" • Response: – "0[d]\r" – "1[d]\r"
Read / EVSE	0x65 (alphabet 'e')	No parameter, 0 bytes	ADC value, 5 characters	'e'	Read ADC value of control pilot	• Command: "e\r" • Example response: "59354[e]\r" ***Note:*** *"59354" is the ADC value in character string format.*
Read / EVSE	0x66 (alphabet 'f')	No parameter, 0 bytes	ADC value, 5 characters	'f'	Read ADC value of proximity pilot	• Command: "f\r" • Example response: "12345[f]\r" ***Note:*** *"12345" is the ADC value in character string format.*
Read / EV	0x67 (alphabet 'g')	No parameter, 0 bytes	PWM duty cycle value, 4 characters	'g'	Read PWM duty cycle in ms (0 – 1000)	• Command: "g\r" • Example response: "0500[g]\r" ***Note:*** *"0500" is the PWM ms value in character string format.*
Read / EV	0x68 (alphabet 'h')	1 byte: • 0: Read 270 Ω resistor. • 1: Read 1.3 kΩ resistor.	Control pilot switch resistor value: • 0: Resistor not set • 1: Resistor set	'h'	Read control pilot switch resistor value	• Command: "h1\r" • Response: "1[h]\r" ***Note:*** *Command byte 2 = '1' indicates 1.3 kΩ CP resistor state request. Response byte 1 = '1' indicates that this resistor is in the ON state.*
Write / EVSE	0x69 (alphabet 'I')	5 character string	None	'I'	Set PWM duty cycle in ms	• Command: "i00500\r" • Response: "[i]\r" ***Note:*** *The command parameter "00500" sets the duty cycle to 50%.*
Write / EVSE	0x6A (alphabet 'j')	No parameter, 0 bytes	None	'j'	Close relay	• Command: "j\r" • Response: "[j]\r"
Write / EVSE	0x6B (alphabet 'k')	No parameter, 0 bytes	None	'k'	Open relay	• Command: "k\r" • Response: "[k]\r"
Write / EV	0x73 (alphabet 's')	2 bytes, byte 2:	None	's'	Set control pilot switch resistors	• Command: "s11\r" • Response: "[s]\r"

Table 16. Supported UART commands*...continued*

Type/side	Command code (sent by host)	Command parameter	Response data (sent by EVSE-SIG-BRD1X)	Response code	Description	Example
		• '0' = Select 270 Ω resistor (not supported). • '1' = Select 1.3 kΩ resistor. byte 3: • '0' = Turn off CP resistor. • '1' = Turn on CP resistor.				***Note:*** *Command bytes 2 and 3 are ASCII coded numbers.* ***Note:*** *For command byte 2; value '0' indicates that 270 Ω resistor is selected, whereas value '1' indicates that 1.3 kΩ resistor is selected.* ***Note:*** *For command byte 3; value '0' indicates that CP resistor has to be turned off, whereas value '1' indicates that CP resistor has to be turned on.*
Read / EVSE, EV	0x76 (alphabet 'v')	No parameter, 0 bytes	Software version number, character string	'v'	Read software version number	• Command: "v\r" • Example response: "01.01.03[v]\r"
Read / EVSE, EV	0x77 (alphabet 'w')	No parameter, 0 bytes	Hardware version number, 1 byte	'w'	Read hardware version number	• Command: "w\r" • Response: – "0[w]\r" for EVSE-SIG-BRD1X board – "2[w]\r" for EVSE-SIG-BRD2X board – and so on
Read / EVSE	0x30 (alphabet '0')	No parameter, 0 bytes	Current voltage and power	'0'	Read current, voltage, and power from meter[1]	• Command: "0\r" • Example response: "4.9 228.5 20.0[0]\r"
Read / EVSE	0x31 (alphabet '1')	No parameter, 0 bytes	Current	'1'	Read current from meter[1]	• Command: "1\r" • Example response: "4.9[1]\r"
Read / EVSE	0x32 (alphabet '2')	No parameter, 0 bytes	Voltage	'2'	Read voltage from meter[1]	• Command: "2\r" • Example response: "228.7[2]\r"
Read / EVSE	0x33 (alphabet '3')	No parameter, 0 bytes	Power	'3'	Read power from meter[1]	• Command: "3\r" • Example response: "20.0[3]\r"
Unknown / EVSE, EV	Any valid command but not supported by the EV/EVSE	Not defined, any number of bytes	Response is 1 byte	'n'	Returns NACK command	• Example command: "x\r" • Response: "[n]\r"

[1] On receiving this command from the host over LPC_UART_RXD (P0_3), EVSE-SIG-BRD1X sends the command to the meter board (TWR-KM35Z75M) over M_UART_TXD (P1_11), requesting for the required parameters. Then, the meter board sends the response to EVSE-SIG-BRD1X through M_UART_RXD (P1_10). Finally, EVSE-SIG-BRD1X sends the response to the host via LPC_UART_TXD (P0_2).

2.10 Host notification

On the EVSE side, the LPC5536/LPC55S36 device uses the following APIs for sending notifications to the host:

- `Advertise_PPStatus()`: Advertises the proximity pilot status to the host whenever the status changes.
- `Advertise_GFCIStatus()`: Advertises the ground fault circuit interrupt status to the host whenever the status changes.
- `Advertise_SleepNotificationToHost()`: Notifies the host before the LPC5536/LPC55S36 MCU enters into Deep-Sleep mode.

After receiving a notification, the host can take an action if required.

All these APIs are called from the `Comm_Process_Advertisement()` API, which, along with the three APIs, is defined in the `evsesigbrd\EVSESigBrdSW\source\commport\comm_command_proc.c` file.

The following is the code snippet for calling the `Advertise_PPStatus()` and `Advertise_GFCIStatus()` APIs:

```
void Comm_Process_Advertisement(void)
{

  /*Check for asynchronous response */
  if((uartRxPortStatus[UART_CONTROL_INDEX] == UART_IDLE) &&
  (uartTxPortStatus[UART_CONTROL_INDEX] == UART_IDLE))
  {
         /*If the counter set by user to put the device into sleep mode reaches
  '0'*/
   if(g_sleepTimeout == 0)
   {
         ....................................
         ....................................

   }

   else if(g_advertiseGFCIState == true)
   {
    g_advertiseGFCIState = false;
    Advertise_GFCIStatus();
   }

   else if(g_advertisePPState == true)
   {
    g_advertisePPState = false;
    Advertise_PPStatus();
   }

   ....................................
   ....................................


  }

 }
```

Use of the `Advertise_SleepNotificationToHost()` API is explained in Section 2.12.1.2.

2.11 Meter notification

On the EVSE side, the LPC5536/LPC55S36 device uses another API, `SendChargingStatusToMeter()`, for sending a notification to the meter board (TWR-KM35Z75M) over the auxiliary UART port (`M_UART_TXD` signal). After receiving the notification, the meter board sets power (P) and current (I) to zero if no charging session is ongoing.

The `SendChargingStatusToMeter()` API is called from the `Comm_Process_Advertisement()` API, as shown below:

```
void Comm_Process_Advertisement(void)
{

  /*Check for asynchronous response */
  if((uartRxPortStatus[UART_CONTROL_INDEX] == UART_IDLE) &&
  (uartTxPortStatus[UART_CONTROL_INDEX] == UART_IDLE))
  {
         /*If the counter set by user to put the device into sleep mode reaches
  '0'*/
   if(g_sleepTimeout == 0)
   {
         ....................................
         ....................................
   }
   ....................................
   ....................................

   else if(g_sendUpdateToMeter == true)
   {
    g_sendUpdateToMeter = false;
    Send_ChargingStatusToMeter();
   }

  }

 }
```

2.12 Power management with Deep-Sleep mode

If the device (LPC5536/LPC55S36 MCU) is inactive for a predefined time, it switches into a low-power mode to reduce power consumption. One such low-power mode is Deep-Sleep mode, which is described in the subsection that follows.

2.12.1 Power modes

The device supports various power modes, each offering a different level of power consumption. By default, the device is in Active mode, where it is fully operational. Even in Active mode, power consumption can be optimized by selectively disabling (either temporarily or permanently) the peripherals that are not in use currently. After any system reset (regardless of the cause), the device always boots into Active mode.

Out of the available low-power modes, Deep-Sleep mode is best suited for the current use case. Here, the device is not completely inactive; some of its functions are active that need power.

2.12.1.1 Deep-Sleep mode

To reduce power consumption significantly, the device uses Deep-Sleep mode. In this mode, analog peripherals do not consume any power. Moreover, the CPU, its memory system and associated controllers, and internal buses do not consume any dynamic power. In other words, Deep-Sleep mode suspends most device operations. Therefore, this mode is ideal for times when the device is not actively communicating with the host.

2.12.1.2 Entering Deep-Sleep mode

On the EVSE side, the LPC5536/LPC55S36 device transitions to Deep-Sleep mode when no communication happens with the host for a predefined time duration. This time duration is specified using the `DEEP_SLEEP_TIMEOUT` macro, which is defined in the `sigbrd_application.h` file. The user can reconfigure this macro to specify a suitable timeout duration for a given application.

The following is the code snippet for entering Deep-Sleep mode:

```
if(g_SleepNotification == SLEEP_NOTIFICATION_SENT)
  {
   g_sleepTimeout = DEEP_SLEEP_TIMEOUT;
   g_SleepNotification = SLEEP_NOTIFICATION_DISABLED;
   g_uartRxTimeout = DEMO_UART_RX_PORT_TIMEOUT;
   g_MetrologyDelayCounter = HUN_MILLI_SEC;
   g_SafetyDelayCounter = TWO_SEC;
   g_SafetyLibRunTimeFlag = false;

   /*Turn off the LED1 & LED2 before entering into sleep mode*/
      GPIO_PinWrite(GPIO, BOARD_LED1_PORT, BOARD_LED1_PIN, 1u);
```

```
     GPIO_PinWrite(GPIO, BOARD_LED2_PORT, BOARD_LED2_PIN, 1u);

   /*Entering the device into Deep-Sleep mode*/
   POWER_EnterDeepSleep(g_excludeFromDS, 0x0, g_wakeupFromDS, 0x0);
  }
```

The whole process can be divided into the following steps:

1. Tracking of the time since the last communication with the host: When the UART port becomes idle (no TX or RX activity at the `LPC_UART_TXD` or `LPC_UART_RXD` signal), an internal variable (sleep timeout counter) `g_sleepTimeout` starts tracking the time left for the device to enter Deep-Sleep mode.
2. Timeout reached: When `g_sleepTimeout` reaches zero and both the `LPC_UART_TXD` and `LPC_UART_RXD` signals are in Idle state, the device notifies the host that it is about to enter Deep-Sleep mode. The notification is sent through the `Advertise_SleepNotificationToHost()` API, which is called from the `Comm_Process_Advertisement()` API, as shown below:

```
void Comm_Process_Advertisement(void)
{

 /*Check for asynchronous response */
 if((uartRxPortStatus[UART_CONTROL_INDEX] == UART_IDLE) &&
 (uartTxPortStatus[UART_CONTROL_INDEX] == UART_IDLE))
 {
        /*If the counter set by user to put the device into sleep mode
 reaches '0'*/
  if(g_sleepTimeout == 0)
  {
   if(g_SleepNotification == SLEEP_NOTIFICATION_DISABLED)
   {
    /*Advertise to host before entering into deep sleep mode*/
    Advertise_SleepNotificationToHost();
    g_SleepNotification = SLEEP_NOTIFICATION_ENABLED;
   }

   else if(g_SleepNotification == SLEEP_NOTIFICATION_ENABLED)
   {
    /*Sleep Notification sent*/
    g_SleepNotification = SLEEP_NOTIFICATION_SENT;
   }
  }

  ....................................
  ....................................

 }

}
```

3. Final UART check: After sending the notification, the device performs a final check on the status of the UART TX port. It ensures that any ongoing transmissions are completed before entering Deep-Sleep mode.
4. Deep-Sleep mode initiation: If the UART ports are confirmed to be idle, the device:
   a. Updates the values of some variables.
   b. Turns OFF both LED1 (D18) and LED2 (D19).
   c. Enters into Deep-Sleep mode by calling the `POWER_EnterDeepSleep()` API from the main program.

2.12.1.3 Wake up from Deep-Sleep mode

On the EVSE side, one of the following methods can be used to wake up the LPC5536/LPC55S36 device from Deep-Sleep mode:

- **Using a UART command from the host**: Any valid UART command received from the host triggers a wake-up event, bringing the device out of Deep-Sleep mode.
- **Using a proximity pilot signal**: An external proximity pilot signal can initiate a wake-up sequence (through the LPC5536/LPC55S36 MCU pin PIO0_28) bringing the device out of Deep-Sleep mode. LED2 (D19) starts blinking whenever the device wakes up from Deep-Sleep mode.

2.13 CAN PHY

EVSE-SIG-BRD1X has a controller area network (CAN) PHY (NXP TJA1044GT) to communicate with other devices in EV/automotive use case. Alternatively, the board can be used as a host controller UART-to-CAN bridge in a standalone use case.

2.13.1 Schematic design

Figure 42 shows the CAN PHY schematic design.

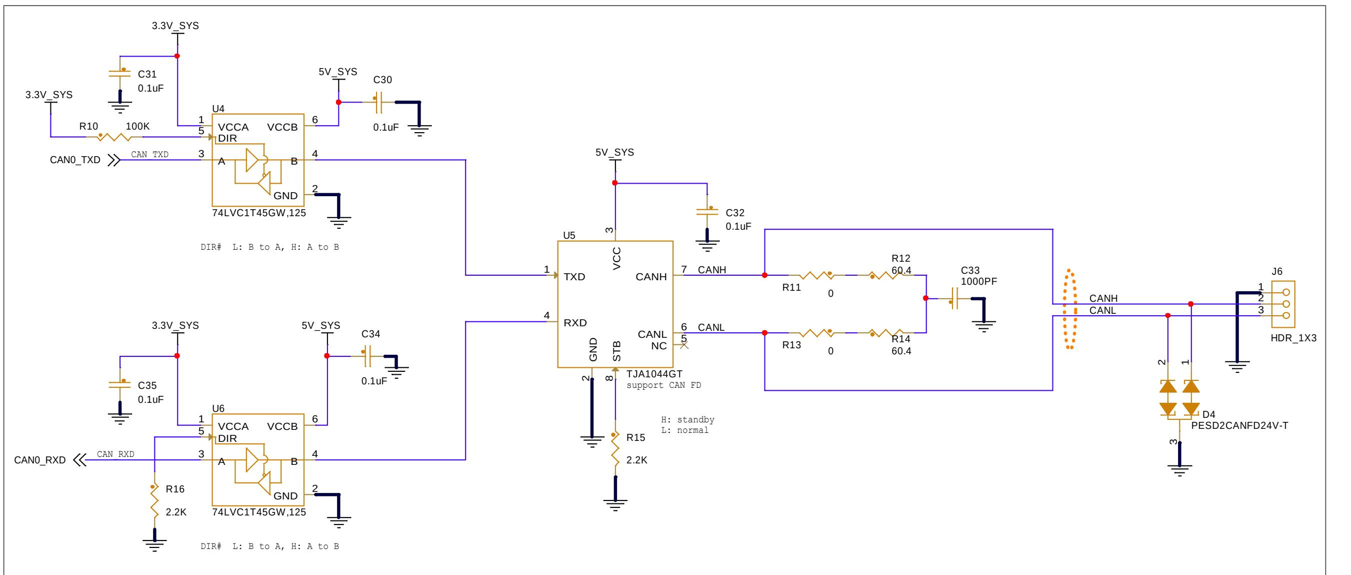

Figure 42. CAN PHY schematic design

2.14 LIN PHY

The multi-function port (MFP) supports local interconnect network (LIN) and serial peripheral interface (SPI) interfaces. EVSE-SIG-BRD1X has a LIN PHY (NXP TJA1021T), which allows the LPC5536/LPC55S36 MCU to communicate with an S32G-VNP-RDB2 or S32G-VNP-RDB3 development board through the LIN interface.

Note: *The LIN PHY can be configured as master by populating the following components:*

- *0 Ω resistor R19 and 1 kΩ resistor R24*
- *Schottky diode D6*

2.14.1 Schematic design

Figure 43 shows the LIN PHY schematic design.

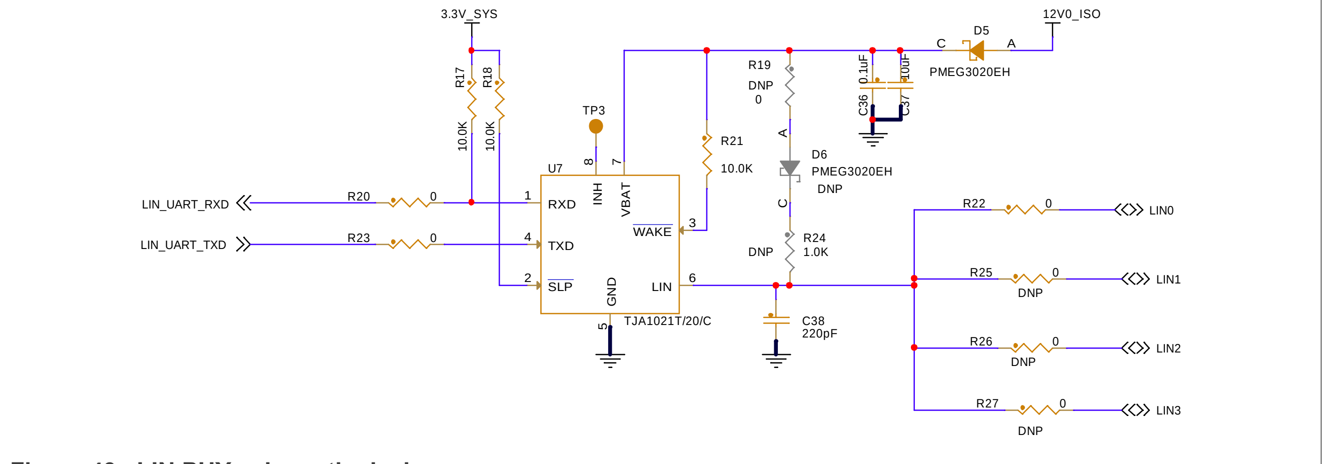

Figure 43. LIN PHY schematic design

By default, the LIN0 pin of the MFP connector is routed through R22. To use any other LIN port, remove R22 and populate the corresponding resistor (R25, R26, or R27) on the board.

2.15 Host connectors

EVSE-SIG-BRD1X provides several options for connecting different host processor development boards. For example, a host processor board can be connected via a SPI connection to the HPGP or a UART to the onboard LPC5536/LPC55S36 MCU. The supported host connections are listed below:

- Arduino socket connectors, which fetch power from the host board and support SPI and UART connections. The Arduino socket is used to connect EVSE-SIG-BRD1X to a development board for an i.MX RT crossover MCU (for example, i.MX RT1060 EVK) or an S32K3 automotive MCU (S32K3X4EVB-T172).
- EXP CN connector, which is an alternative to the Arduino socket. It connects seamlessly with an i.MX evaluation kit (EVK) board, such as i.MX 8M Nano EVK or MCIMX93-EVK.
- Multi-function port (MFP) connector, which connects seamlessly with an S32G-VNP-RDB2 or S32G-VNP-RDB3 development board.

Table 17, Table 18, Table 19, and Table 20 show the pinouts of the Arduino connectors J40, J36, J37, and J39, respectively.

Table 17. Arduino connector J40 pinout

Pin numbers	Signal name	Type	Description
1	HOST_UART_TXD	O	Host UART transmit
8	HOST_UART_TXD (disconnected)		
2	HOST_UART_RXD	I	Host UART receive
7	HOST_UART_RXD (disconnected)		
3, 4, 5, 6			Not connected

Table 18. Arduino connector J36 pinout

Pin numbers	Signal name	Type	Description
2	HOST_SPI_CS1	I	Host SPI master chip select option 1

Table 18. Arduino connector J36 pinout*...continued*

Pin numbers	Signal name	Type	Description
3	HOST_SPI_CS2	I	Host SPI master chip select option 2
4	HOST_SPI_MOSI	I	Host SPI master output slave input signal
5	HOST_SPI_MISO	O	Host SPI master input slave output signal
6	HOST_SPI_CLK	I	Host SPI master clock
7	Ground		Ground
1, 8, 9, 10			Not connected

Table 19. Arduino connector J37 pinout

Pin numbers	Signal name	Type	Description
4	3V3_ARD_EXP_CN	Power	+3.3 V
5	5V_ARD_EXP_CN	Power	+5 V
6, 7	Ground		Ground
1, 2, 3, 8			Not connected

Table 20. Arduino connector J39 pinout

Pin numbers	Signal name	Type	Description
1	HPGP_GP_IRQ	O	HPGP general-purpose interrupt
3	HOST_SPI_IRQ	O	HPGP_SPI_IRQ || SW_INT_N (selection through J43): • HPGP_SPI_IRQ: HPGP SPI interrupt • SW_INT_N: SJA1110B switch interrupt
4	HPGP_RESET	I	HPGP reset signal
2, 5, 6			Not connected

Table 21 shows the pinout of the EXP CN connector J44.

Table 21. EXP CN connector J44 pinout

Pin numbers	Signal name	Type	Description
1, 17	3V3_ARD_EXP_CN	Power	+3.3 V
2, 4	5V_ARD_EXP_CN	Power	+5 V
7	HOST_SPI_IRQ	O	HPGP_SPI_IRQ || SW_INT_N (selection through J43): • HPGP_SPI_IRQ: HPGP SPI interrupt • SW_INT_N: SJA1110B switch interrupt
8	HOST_UART_RXD	I	Host UART receive
10	HOST_UART_TXD	O	Host UART transmit
11	HPGP_GP_IRQ	O	HPGP general-purpose interrupt
12	HPGP_RESET	I	HPGP reset signal

Table 21. EXP CN connector J44 pinout...*continued*

Pin numbers	Signal name	Type	Description
16	HOST_SPI_CS2 (disconnected)	I	Host SPI master chip select option 2
26	HOST_SPI_CS2		
19	HOST_SPI_MOSI	I	Host SPI master output slave input signal
21	HOST_SPI_MISO	O	Host SPI master input slave output signal
23	HOST_SPI_CLK	I	Host SPI master clock
24	HOST_SPI_CS1	I	Host SPI master chip select option 1
6, 9, 14, 20, 25, 30, 34, 39	Ground		Ground
3, 5, 13, 15, 18, 22, 27, 28, 29, 31, 32, 33, 35, 36, 37, 38, 40			Not connected

Table 22 shows the pinout of the MFP connector J45.

Table 22. MFP connector J45 pinout

Pin numbers	Signal name	Type	Description
1	5V_ARD_EXP_CN	Power	+5 V
3	3V3_ARD_EXP_CN	Power	+3.3 V
9	LIN1	I/O	LIN master 1
10	LIN0	I/O	LIN master 0
11	LIN3	I/O	LIN master 3
12	LIN2	I/O	LIN master 2
13	HPGP_GP_IRQ	O	HPGP general-purpose interrupt
14	HOST_SPI_IRQ	O	HPGP_SPI_IRQ || SW_INT_N (selection through J43): • HPGP_SPI_IRQ: HPGP SPI interrupt • SW_INT_N: SJA1110B switch interrupt
15	HOST_SPI_MISO	O	Host SPI master input slave output signal
16	HOST_SPI_MOSI	I	Host SPI master output slave input signal
17	HOST_SPI_CS1	I	Host SPI master chip select option 1
18	HOST_SPI_CLK	I	Host SPI master clock
5, 6, 7, 19, 20, 21	Ground		Ground
2, 4, 8, 22, 23, 24, 25, 26			Not connected

3 Appendix A

Figure 44 shows the cross-section of an AC EV charging Type 2 connector.

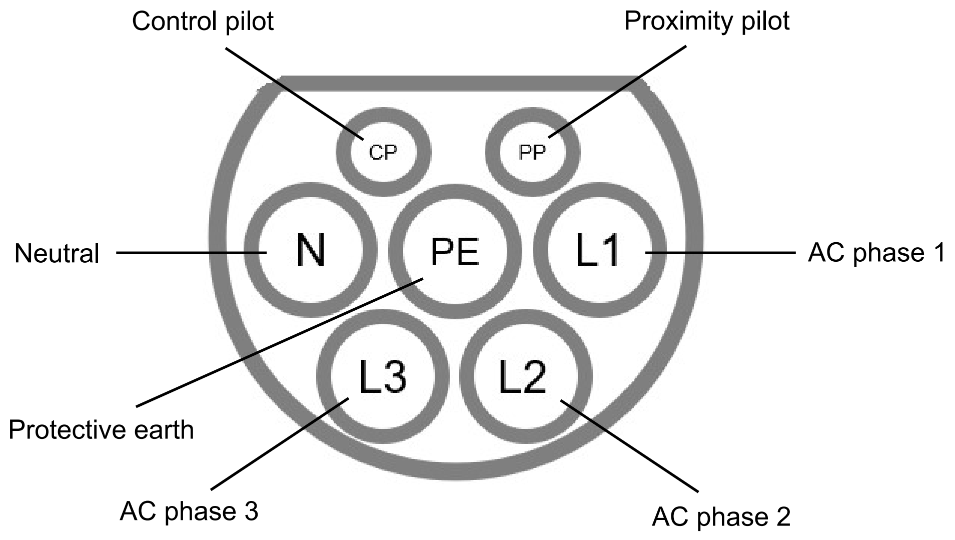

Figure 44. AC EV charging Type 2 connector

4 Related resources

Table 23 lists some additional resources that may be required while working on EVSE-SIG-BRD1X.

Table 23. Related resources

Resource	Link / how to obtain
EVSE-SIG-BRD1X User Guide (UG10109)	Contact an NXP field applications engineer (FAE) or sales representative
SAE Electric Vehicle and Plug in Hybrid Electric Vehicle Conductive Charge Coupler (J1772_202401)	https://www.sae.org/standards/content/j1772_202401/
Lumissil website (connectivity)	https://www.lumissil.com/products/wired-communication
Lumissil customer support portal	https://cbu-support.lumissil.com/
Lumissil support email address	connectivity_support@lumissil.com

5 Acronyms

Table 24 lists the acronyms used in this document.

Table 24. Acronyms

Acronym	Description
AC	Alternating current
ADC	Analog-to-digital converter
AFE	Analog front-end
ASIC	Application-specific integrated circuit
CAN	Controller area network
CP	Control pilot
CRC	Cyclic redundancy check
CS	Chip select
CT	Current transformer
DAC	Digital-to-analog converter
DC	Direct current
DIP	Dual inline package
DS	Design Studio
eFlexPWM	Enhanced Flexible Pulse Width Modulator
EV	Electric vehicle
EVSE	Electric vehicle supply equipment
FD	Flexible data-rate
FlexCAN	Flexible controller area network
Flexcomm	Flexible serial communication
FlexPWM	Flexible pulse width modulator
FlexSPI	Flexible serial peripheral interface
FS	Full-speed
GFCI	Ground fault circuit interrupter
GPIO	General-purpose input/output
HPGP	HomePlug Green PHY
IP	Internet protocol
ISO	International Organization for Standardization
ISP	In-System Programming
JTAG	Joint Test Action Group
LED	Light-emitting diode
LIN	Local interconnect network
MCU	Microcontroller unit
MFP	Multi-function port

Table 24. Acronyms*...continued*

Acronym	Description
MII	Media-independent interface
MISO	Master input slave output
MOSI	Master output slave input
NFC	Near-field communication
OFDM	Orthogonal frequency division multiplexing
Op-amp	Operational amplifier
OTP	One-time-programmable
PER	Packet error rate
PHY	Physical layer
PLC	Power-line communication
PP	Proximity pilot
PWM	Pulse width modulator
QEI	Quadrature encoder interface
QFN	Quad flat no-lead
QSPI	Quad serial peripheral interface
RGMII	Reduced gigabit media-independent interface
RMII	Reduced media-independent interface
RTC	Real-time clock
SDK	Software development kit
SDP	SECC Discovery Protocol
SECC	Supply Equipment Communication Controller
SGMII	Serial gigabit media-independent interface
SLAC	Signal Level Attenuation Characterization
SMA	SubMiniature version A
SPDT	Single pole double through
SPI	Serial peripheral interface
SRAM	Static random-access memory
SWD	Serial wire debug
SWO	Serial wire debug trace output
TCP	Transmission control protocol
TDI	Test data in
TMS	Test mode select
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous/asynchronous receiver/transmitter
USB	Universal serial bus

6 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2024 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7 Revision history

Table 25 summarizes the revisions to this document.

Table 25. Revision history

Document ID	Release date	Description
UM12013 v.1	18 June 2024	Initial public release

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

i.MX — is a trademark of NXP B.V.

Kinetis — is a trademark of NXP B.V.

Contents

1 Board overview 2
1.1 Block diagrams 2
1.2 Board features 4
1.3 Board pictures 5
1.4 Connectors 6
1.5 Jumpers 7
1.6 Push button and DIP switch 10
1.7 LEDs 12
2 Functional description 14
2.1 Power supplies 14
2.1.1 Block diagram 14
2.1.2 Power supply sources 15
2.1.3 Schematic design 16
2.1.3.1 PMIC PPF5020 17
2.1.3.2 Boost converter and charge pump inverter 18
2.2 Clocks 19
2.3 Proximity pilot 19
2.3.1 Schematic design 20
2.3.2 Software implementation 21
2.4 Control pilot 22
2.4.1 J1772 PWM 24
2.4.1.1 Schematic design 25
2.4.1.2 eFlexPWM usage for control pilot for EVSE 27
2.4.1.3 Software implementation of eFlexPWM 28
2.4.1.4 CTIMER usage for control pilot 29
2.4.1.5 Software implementation of CTIMER 29
2.4.1.6 ADC usage for control pilot for EVSE 31
2.4.1.7 Software implementation of ADC 31
2.4.2 HomePlug Green PHY 32
2.4.2.1 Host interface 33
2.4.2.2 CG5317 bootstrap 35
2.4.2.3 CG5317 UART debug port 36
2.4.2.4 CG5317 analog interface 36
2.5 GFCI circuit 36
2.5.1 Schematic design 36
2.5.2 Software implementation 37
2.6 Relay driver circuit 38
2.6.1 Block diagram 38
2.6.2 Schematic design 38
2.6.3 Software implementation 39
2.7 LPC5536/LPC55S36 MCU 40
2.7.1 Block diagram 41
2.7.2 Schematic design 41
2.7.3 LPC5536/LPC55S36 pin usage 42
2.7.4 LPC5536/LPC55S36 boot options 44
2.7.5 Debug interface 44
2.8 SJA1110B switch 45
2.8.1 Block diagram 45
2.8.2 Schematic design 46
2.8.3 SJA1110B pin usage 47
2.8.4 SJA1110B bootstrap 49
2.8.5 SJA1110B software 49
2.8.6 Debug interface 49
2.9 UART interface 49
2.9.1 Software implementation 50
2.9.2 UART commands 51
2.10 Host notification 53
2.11 Meter notification 54
2.12 Power management with Deep-Sleep mode 55
2.12.1 Power modes 55
2.12.1.1 Deep-Sleep mode 55
2.12.1.2 Entering Deep-Sleep mode 55
2.12.1.3 Wake up from Deep-Sleep mode 57
2.13 CAN PHY 57
2.13.1 Schematic design 57
2.14 LIN PHY 57
2.14.1 Schematic design 57
2.15 Host connectors 58
3 Appendix A 61
4 Related resources 62
5 Acronyms 63
6 Note about the source code in the document 65
7 Revision history 66
Legal information 67

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

© 2024 NXP B.V. All rights reserved.

For more information, please visit: https://www.nxp.com